General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

NSG-7506

UNIVERSITY OF ILLINOIS
URBANA

AERONOMY REPORT NO. 115

FORTH SYSTEM FOR COHERENT-SCATTER RADAR DATA ACQUISITION AND PROCESSING

by
A. D. Rennier
S. A. Bowhill

February 1, 1985

Library of Congress ISSN 0568-0581

Aeronomy Laboratory
Department of Electrical and Computer Engineering
University of Illinois
Urbana, Illinois

Supported by
National Aeronautics and Space Administration

UILU-ENG-85-2501

A E R O N O M Y R E P O R T

N O. 115

FORTH SYSTEM FOR COHERENT-SCATTER RADAR DATA ACQUISITION AND PROCESSING

by

A. D. Rennier
S. A. Bowhill

February 1, 1985

Supported by
National Aeronautics
and Space Administration
Grant NSG 7506

Aeronomy Laboratory
Department of Electrical and
Computer Engineering
University of Illinois
Urbana, Illinois

ABSTRACT

A real-time collection system was developed for the Urbana coherent-scatter radar system. The new system, designed for use with a micro-computer, has several advantages over the old system implemented with a minicomputer. This work describes the software used to collect the data as well as the processing software used to analyze the data. In addition a magnetic tape format for coherent-scatter data exchange is given.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Coherent Scatter Technique

Winds and waves in the upper atmosphere can be observed through the use of radar. As a VHF pulse propagates through the atmosphere it encounters discontinuities in refractive index thought to be caused by winds and waves. Part of its energy is then reflected back to the ground, and is detected by a ground-based antenna and receiver. The altitude of the discontinuity can be determined from the time elapsed between transmission and reception of the signal. By examining the returned signal it is also possible to determine some properties of the discontinuity that caused the signal to return. For example, by measuring the power in the returned signal the strength of the discontinuity can be determined; and by comparing the frequency of the returned signal with that of the transmitted signal a Doppler frequency can be measured. This Doppler frequency is proportional to the line-of-sight velocity of the discontinuity. By monitoring the returned signal over a period of time, it can be determined how fast the discontinuity is changing, since the rate of change is inversely proportional to the correlation time of the signal.

1.2 Urbana System

For a complete description of the Urbana system see Gibbs and Bowhill (1983). A brief summary of the Urbana system is given here.

At Urbana the coherent scatter technique is used to determine the power, velocity and correlation time of the signals returned from the atmosphere. The system used consists of a transmitter and receiver both connected to a phased antenna array via a T/R (transmit/receive) switch. The transmitter sends a 40.92 MHz, 10-microsec pulse into the atmosphere every

2.5 millisec. After a pulse the antenna is switched over to the receiver so that the returned signal can be monitored. The received signal is coherently detected; that is, it is resolved into its real and imaginary parts in the receiver. This is done by multiplying the returned signal by the transmitted signal (real) and by the transmitted signal shifted by 90 degrees (imaginary). These two signals are then fed into an interface with two sample and hold circuits. The signals are converted to digital form every 10 microsec giving an altitude resolution of 1.5 km in the atmosphere. The microcomputer is then used to collect and integrate the data into 1/8 sec samples. The 1/8 sec samples are then used to calculate information for an autocovariance function. This information is then averaged for one minute and stored on disk. One hour of data may be stored on one side of a floppy disk. The disks are then later analyzed to determine the one-minute averages of the power, velocity and correlation time of the data.

The power is calculated from the zeroth lag of the autocovariance function.

$$\text{Power} = 20\ \text{Log}_{10}\sqrt{2RR^2(0) + 2II^2(0)}$$

$$RR(0) = \sum_{i=1}^{400} |R_i|$$

$$II(0) = \sum_{i=1}^{400} |I_i|$$

The velocity is determined from the real and imaginary components of the autocovariance function (Bowhill 1983).

$$\text{Velocity} = \frac{\lambda(\omega\tau)}{4\pi\tau}$$

$$\omega\tau = \text{Arctan}\left(\frac{\text{imaginary component}}{\text{real component}}\right)$$

λ = wavelength

τ = time between samples (1/4 or 1/8 sec)

Note that the arithmetic rather than the geometric mean is used in the calculation of the real component.

The correlation time is the time required for the magnitude of the autocovariance to fall to 1/2 its value at time equal zero. Assuming that the magnitude of the autocovariance function is Gaussian the correlation time is given by

$$\text{Correlation time} = k\sqrt{\frac{1}{\ln(\rho_1/\rho_2)}}$$

K = constant $\quad t_1 = 1/8$ sec

$\rho_1 = \rho_0 e^{-t_1^2/T^2} \quad t_2 = 1/4$ sec

$\rho_2 = \rho_0 e^{-t_2^2/T^2}$

T = correlation time

The one minute averages of the power, velocity and correlation time are stored in the Apple text files on floppy disks. These text files are plotted and may be transferred to magnetic tape if other users request the data.

1.3 Background to Statement of the Problem

Previously at Urbana a PDP-15 minicomputer was used to collect coherent scatter data. With the advent of microcomputers it became possible to configure a smaller system with the same capabilities as the PDP-15. To this end a collection program was written in FORTH for the Apple microcomputer. The new system had several advantages over the old. 1) The Apple was more convenient to use than the PDP-15. 2) The Apple utilized floppy disks rather than magnetic tapes for its mass storage. 3) The Apple had directly addressable screen memory, enabling a real-time display to be updated 8 times a second. Aside from these advantages the Apple had one drawback. The Apple is equipped with a 6502 microprocessor with a

maximum clock rate of 1 MHz. This meant that only one A/D could be read and its value stored in the allotted 10 microsec interval. In order to maintain the 1.5 km spatial resolution in the atmosphere the FORTH program was designed to alternately read the real and imaginary channels. For instance, when the first pulse was transmitted, the Apple would sample only the real channel at all heights. Then when the second pulse was transmitted the Apple would sample the imaginary channel at all the heights. It was determined that with the aid of a booster card in the Apple it was possible to read both the real and imaginary samples in the allotted 10 microsec.

1.4 Statement of Problem

1) Increase the speed of the FORTH collection program so that the real and imaginary channels are sampled simultaneously.

2) Modify, merge and document existing analysis programs to process data taken with the improved collection program.

3) Design and implement a data format for magnetic tapes used to exchange coherent-scatter data with other potential users.

2. COHERENT-SCATTER COLLECTION SYSTEM

In order to supply the analysis progam with the information needed to calculate the quantities discussed in Section 1.2, the collection program was written. The collection program monitors coherent-scatter radar returns and calculates the information needed for the autocovariance function. This information is then stored on disk in the form of one-minute averages.

2.1 Functional Description

2.1.1 Introduction: The collection program is shown in Appendix A and is written in fig-FORTH and Assembly Language for a 6502 microprocessor. It is designed to run on an Apple II microcomputer with a John Bell Engineering Parallel Interface Card, a Scitronics Real-Time Clock Card, an Apple Floppy Disk Drive and a Number Nine Booster Card.

The collection program is used to measure coherent-scatter radar returns from the atmosphere. The data are collected by sending the real and imaginary components of the received signal to an interface box containing two 8-bit A/D converters with sample and hold circuits. The output from the A/D converters is read by the John Bell Card via ribbon cable connections. The John Bell Card is also responsible for sending a 100 KHz start-convert signal to the interface box. The start-convert signal indicates when a new sample should be taken and converted into digital form. The 100 KHz sampling rate gives a spatial resolution of 1.5 km in the atmosphere.

The collection program is interrupt driven. This means that there are two processes that work together in the collection program, the interrupt process and the background process. The background process is the main portion of the program and is used to do all the housekeeping chores.

Initially the background process executes until the Apple detects an interrupt. The interrupt causes the Apple to stop executing the background process and start executing the interrupt process. When the interrupt process is complete the Apple resumes executing the background process at precisely the point where it left off. Execution of the background process continues until another interrupt is detected. The interrupts are generated by the same 400 Hz square wave used to trigger the transmitter. Thus, every time a pulse is transmitted into the atmosphere the interrupt process is executed.

2.1.2 Interrupt Processing: The interrupt process is used to collect and integrate the real and imaginary signals sampled by the interface box. This task is accomplished in the following manner.

1) The X and Y registers in the microprocessor are saved on the stack. This is done so that the background process can continue on after the interrupt process is finished.

2) The interrupt flag is cleared so that the next time an interrupt occurs the Apple is able to detect it.

3) One of the 6522s on the John Bell Card is set up to send the 100 KHz start-convert signal to the interface box.

4) A delay is executed to allow the interrupt process to wait for the desired range of radar returns. The length of the delay is based on the amount of time needed for the transmitted pulse to propagate to the lowest desired altitude and back.

5) Sixty heights of data spaced by 1.5 km are collected. This is done by reading both A/Ds (real and imaginary) every 10 microsec and storing the data in a buffer called INBUFFER.
   INBUFFER is a one page (256 bytes) section of memory divided into two equal parts. The first half is used to store the real component of the data, while the second half is used to store the imaginary component. INBUFFER has the capacity to store 128 heights of data. Since the collection program only collects 60 heights of data the 136 bytes (68 real and 68 imaginary) extra bytes are not used.

6) The contents of INBUFFER are added height by height to the current opening in the RINGBUFFER.
   RINGBUFFER is a four page (1024 bytes) section of memory used to accumulate input data. RINGBUFFER can be viewed as a 16-bit,

three-dimensional array.

RINGBUFFER (QUANTITY, SECTION, HEIGHT)

Where QUANTITY is either real or imaginary, SECTION is an integer between 1 and 4, and HEIGHT is a number between 1 and 64. The array is characterized by 16-bits because 16 bits are allotted each entry in the array. From this description it can be seen that RINGBUFFER has four sections in both the real and imaginary portions of the array. Each of these sections has the capacity to hold 64 heights of 16-bit data. Since only 60 heights of data are collected the remaining four entries in each section are not used.

Each section of RINGBUFFER represents one sample or 1/8 sec of accumulated data (50 pulses). The current opening in RINGBUFFER is that section which is being used to accumulate the present sample. After one section of RINGBUFFER has been filled the interrupt process is directed to begin filling the next section. When the RINGBUFFER is full the interrupt process will reuse the first section, then the second section etc., for the duration of the program.

7) The number of pulses in the current sample is updated. When the correct number of pulses has been collected the interrupt process disables the Apple from detecting any more interrupts until the required background processing is complete.

8) The speaker in the Apple is clicked. The result of this is a steady tone from the Apple when the interrupt process is being called properly. The tone serves as a feedback to the operator that the collection program is working properly.

9) The X and Y registers are restored.

10) The interrupt process completes and returns control of the Apple to the background processing.

2.1.3 Background Processing: The background process performs all the housekeeping functions, calculates the information for the autocovariance function, and writes the one-minute averages of this information on disk. These tasks are accomplished in the following manner.

1) The program is initialized. When the program is started it is necessary to partition and intialize memory before any data can be collected. During the initialization the interrupts are ignored.

2) The program waits for a new minute on the internal clock. To help synchronize program execution the internal clock is monitored in a wait loop for the next minute change. When a minute change is detected, the interrupt is enabled.

3) The program correlates one minute of data. It is in this step that the information for the one-minute averages of the auto-covariance function are accumulated. In order to correlate the data, data from previous samples are needed. So before the one-minute averages are calculated the RINGBUFFER must be preloaded with four samples. Once the preliminary samples are taken the information needed for the autocovariance function is accumulated in OUTBUFFER.

OUTBUFFER is a 12 page (3072 bytes) section of memory that can be viewed as a 24-bit, three-dimensional array.

OUTBUFFER(QUANTITY, LAG, HEIGHT)

QUANTITY is either Real-Real (RR), Imaginary-Imaginary (II), Real-Imaginary (RI) or Imaginary-Real (IR). LAG is an integer between -1 and 2, and HEIGHT is an integer between 1 and 64. Note: only 1 through 60 are used as explained earlier. The array is characterized by 24-bits because 24 bits are allotted each entry in the array.

Besides accumulating the information for the autocovariance function OUTBUFFER is also used to accumulate the DC component of the real and imaginary channels.

OUTBUFFER(RR, -1, HEIGHT) = DC Real
OUTBUFFER(II, -1, HEIGHT) = DC Imaginary

The DC component is calculated by summing all the samples for the real and imaginary channels for one minute and dividing by the number of samples taken (400). Two parts of OUTBUFFER are not used.

OUTBUFFER(RI, -1, HEIGHT) = Not Used
OUTBUFFER(IR, -1, HEIGHT) = Not Used

There are two portions of OUTBUFFER used to accumulate data necessary to the calculation of the power in returned siganl.

OUTBUFFER(RR, 0, HEIGHT) = Absolute Real
OUTBUFFER(II, 0, HEIGHT) = Absolute Imaginary

Absolute Real and Absolute Imaginary are equal to the sum of the absolute values of each sample. The rest of OUTBUFFER is used to accumulate the absolute values of the differences between the combinations of the real and imaginary samples. For instance, if QUANTITY equals RR and, LAG equals 2, then that portion of OUTBUFFER is the sum of the absolute values of the differences between the real sample just collected and the real sample collected 2/8 sec before. If QUANTITY equals IR and LAG equals zero then that portion equals the sum of the absolute values of the difference between the imaginary sample just collected and the real sample just collected.

It is important to remember that there are other things going on while OUTBUFFER is being filled. The whole time that the data

are being correlated the background process is being interrupted and new data is being put into the RINGBUFFER. When a section of RINGBUFFER is filled it is the job of background processing to advance the queue and display on the Apple monitor the lowest 20 altitudes of the real and imaginary samples just taken. In addition, the background process puts the negative of the DC component into the next available portion of RINGBUFFER. The reasons for this are explained in step 4.

There are 400 samples taken to represent each minute of data. The 400 1/8 sec samples account for approximately 50 sec of data, the remaining 10 sec being used to do other housekeeping and to access the disk.

4) The DC components of the real and imaginary samples are calculated. In order to eliminate ground clutter and be consistent with the processing algorithm the background process calculates the DC value for each channel, normalizes the value to one sample and inserts the negative of the normalized value into each new portion of RINGBUFFER before any data is added in. After each minute the DC values are modified to accommodate the changes in the received signal.

Notice that steps 2-4 are repeated in steps 5-11. Steps 2-4 are used to collect data for a dummy minute. The dummy minute is used only for the calculation of the DC components, the rest of the information in the dummy minute is not used.

5) OUTBUFFER is filled with zeros. Since OUTBUFFER is an accumulation buffer it is necessary to empty it between successive minutes.

6) The program correlates another minute of data. This step is identical to step 3, except that this time a non-zero DC component has been subtracted out of the samples.

7) The program updates the DC estimate. The reasons for this are explained in step 4.

8) The program displays the amplitude and the frequency (A,F) of the lowest 20 altitudes of the signal for the past minute. This information is displayed on the Apple monitor until the minute change.

9) The program saves OUTBUFFER to another location. There are two situations possible here. If the first, third or any other odd minute has just completed the middle and high bytes of OUTBUFFER are moved to another location in memory. If the second, fourth or any even minute has completed the medium and high bytes of this minute along with the medium and high bytes of the previous odd minute are saved on disk in one binary file.

10) The amplitude and frequency indicators (A,F) on the Apple monitor are erased.

11) The background process jumps back to step 5 and continues executing.

2.2 Technical Description

In order to describe precisely how the collection program performs its tasks, it is necessary to have a technical description of the program. In this section there is a complete listing of all the definitions used in the collection program, as well as an explanation to their use. A textbook on FORTH (Scanlon 1982) and a copy of the fig-FORTH system (Lyons 1981) are essential to the understanding of the collection program. A copy of the collection program is found in Appendix A, which should be referred to in reading the following text.

The Apple clock has a period of 1.052 microsec. For simplicity all times given in the following text are based on an Apple clock with a one microsec period.

There are constants and variables included in the following definitions. It is important to remember that variables may be changed after the program is compiled. If a constant needs to be changed the program must be modified and recompiled.

INPUTSLOT (screen 32) is the constant used to indicate which slot in the Apple has the John Bell Card. The slots in the Apple are numbered 0-7 left to right.

CLOCKSLOT (screen 32) is the constant used to indicate which slot in the Apple has the internal clock card.

INBUFFER (screen 32) is the constant used to hold the address of the first byte of the buffer called INBUFFER. All of the buffers in the collection program are located contiguously in memory. When there is a change in the constant INBUFFER all of the buffers in the collection program are moved. A memory map based on INBUFFER equal to 7200H is given in Figure 2.1.

MEMORY MAP

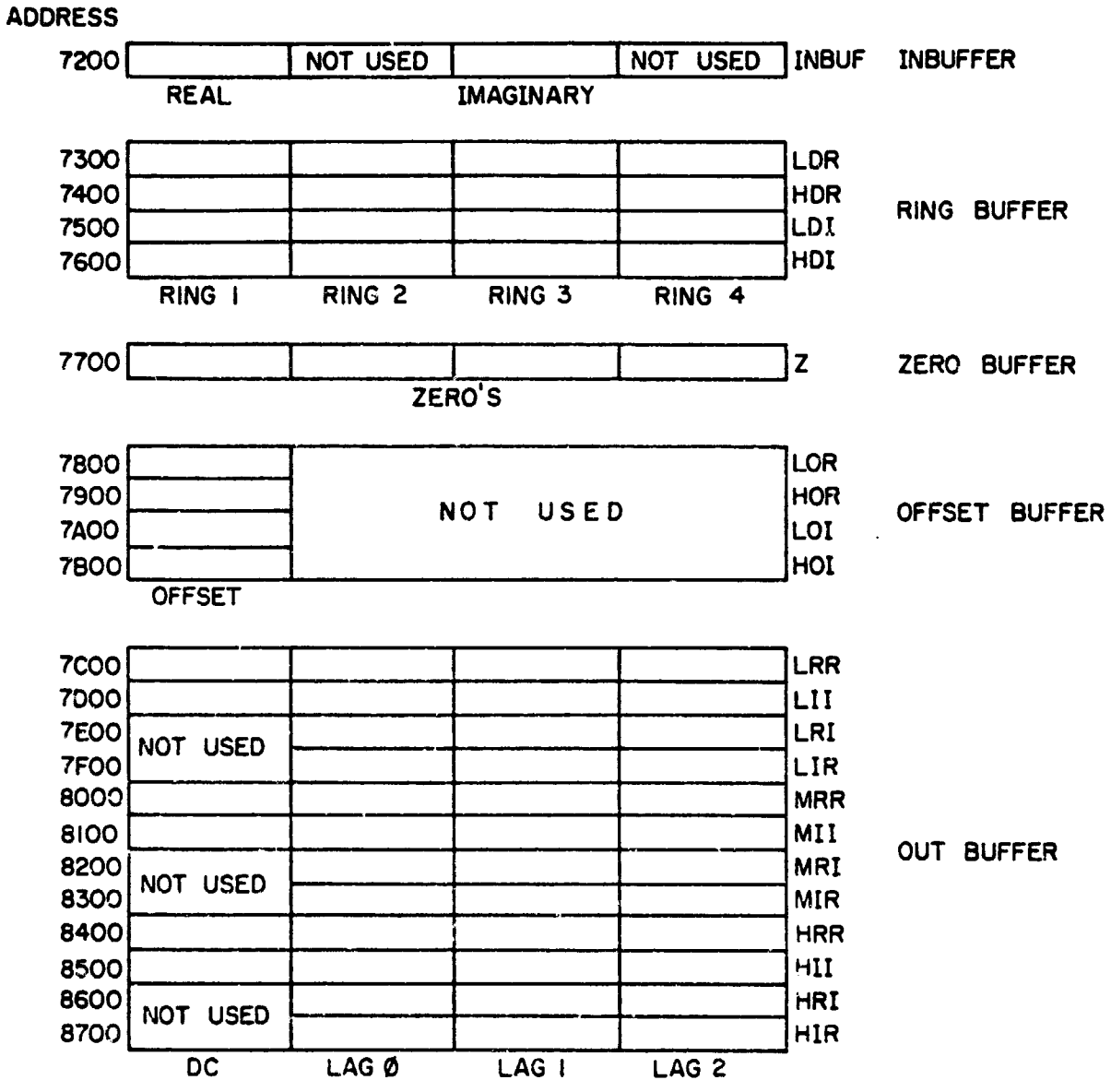

Figure 2.1 Memory map for collection program buffers.

SECTION (screen 32) is a constant used to hold the maximum number of heights that may be stored in each section of RINGBUFFER and OUTBUFFER. Since the amount of memory allotted to the buffers is fixed, SECTION also determines the number of sections in RINGBUFFER and OUTBUFFER. For instance, if 40H maximum heights are desired then 100H/40H or 4 different sections are used in the buffers. 8H maximum heights leave 100H/8H or 20H sections.

LAGS (screen 32) is a constant used to hold the number of previous samples that the collection program holds at any one time. LAGS is equal to the number of sections in RINGBUFFER minus two. This relationship stems from the fact that at any one time RINGBUFFER has one section being filled and another being used to hold the present sample. The rest of the sections in RINGBUFFER are then left to hold previous samples.

HEIGHTS (screen 32) is a constant used to hold the number of heights that the program collects. HEIGHTS can be any number from one to SECTION.

DISPHTS (screen 32) is a constant used to hold the number of heights of data that the collection program displays.

LOWHEIGHT (screen 32) is a constant equal to the value of the lowest height to be displayed. LOWHEIGHT is used to determine the altitude labels that appear on the left side of the Apple screen. The labels displayed on the screen are always the lowest heights collected.

MINS/DISK (screen 32) is a constant used to determine how many minutes of data are stored on one side of a 5 1/4" floppy disk.

SAMPLES (screen 32) is a variable used to hold the number of samples which are collected for the one-minute averages.

PULSES (screen 32) is a variable used to hold the number of transmitter pulses which are integrated to form one sample.

ter pulses which are integrated to form one sample.

MINS (screen 32) is a variable used for test purposes to hold the number of minutes the program runs before ending. If MINS is set equal to or higher than MINS/DISK the verb WRITEFILE resets MIN# after MINS/DISK minutes have executed. If this happens then MINS is effectively ignored.

DELAY1 (screen 32) is a variable used to fine tune the timing of the verb INTERRUPT. DELAY1 is used to precisely adjust the latch and start-convert signals.

DELAY2 (screen 32) is a variable used to tune the timing of the verb INTERRUPT. An increase of one in DELAY2 tells INTERRUPT to wait an additional 5 microsec before reading and unlatching the data.

MIN# (screen 32) is a variable which contains the number of minutes that data has been collected since a new disk was started.

AMPFACTOR (screen 32) is a variable used to scale the one-minute average amplitudes displayed on the Apple screen.

FREQFACTOR (screen 32) is a variable used to scale the one-minute average frequencies displayed on the Apple screen.

INADDR (screen 33) is the constant used as a reference point for defining all of the constants used in addressing the John Bell Card. Each slot in the Apple has one page of addresses associated with it. The address range for a particular slot is CX00H-CXFFH, where X is the slot number. The John Bell Card has two 6522 Versatile Interface Adapters (VIAs) on it. CX00H-CX0FH is used to address the registers of the first VIA, while CX80H-CX8FH is used to address the second VIA. The constants used in addressing the VIAs on the John Bell Card are named below. For a more detailed description of the registers and their functions see Scanlon (1980).

DDRB1-Data direction register B, first VIA.
DDRA1-Data direction register A, first VIA.
ACR1-Auxiliary control register, first VIA.
PCR1-Peripheral control register, first VIA.
IER1-Interrupt enable register, first VIA.
DRB2-Data register B, second VIA.
DDRB2-Data direction register B, second VIA.
T1CL2-Timer 1 counter, low byte, second VIA.
T1CH2-Timer 1 counter, high byte, second VIA.
T1LL2-Timer 1 latch, low byte, second VIA.
T1LH2-Timer 1 latch, high byte, second VIA.
ACR2-Auxiliary control register, second VIA.
PCR2-Peripheral control register, second VIA.
IER2-Interrupt enable register, second VIA.

WINDOW (screen 34) is a constant used to initialize the Apple screen. WINDOW and WINDOW+1 are the zero-page locations used to set the top and bottom boundaries of the text screen.

CURSOR (screen 34) is a constant used to position the cursor on the Apple screen. CURSOR and CURSOR+1 are the zero-page addresses used to set the horizontal and vertical positions of the cursor.

ACC (screen 34) is a constant and is the address used by the Apple monitor to store the value of the microprocessor accumulator when an interrupt is serviced.

IRQVECTORADDR (screen 34) is a constant and is the address of the two-byte Apple interrupt vector, i.e., where execution will jump when an interrupt occurs.

SPEAKER (screen 34) is a constant and holds the address of a latch connected to the Apple's internal speaker. Whenever this address is read the latch toggles and the speaker produces a "click".

QUEUE1 (screen 34) is a constant, the zero-page address of the pointer that keeps track of which section of RINGBUFFER is currently being correlated with the section most recently acquired.

QUEUE2 (screen 34) is a constant, the zero-page address of the pointer that keeps track of which section in RINGBUFFER is being used to calculate

LAG 1.

HREAL and HIMAG (screen 34) are constants and hold the zero-page addresses where the low bytes of two two-byte pointers are located. These pointers are used to pick out the QUANTITY and SECTION of RINGBUFFER that are currently being used for the real time display.

LSUM and HSUM (screen 34) are constants, equal to the zero-page addresses where the low bytes of two two-byte pointers are located. The low bytes of these pointers are used to indicate which section of RINGBUFFER is currently being filled. The high bytes of these pointers indicate whether data are being summed into the real or imaginary part of RINGBUFFER. Two pointers are needed because RINGBUFFER is a two-byte buffer.

LAG (screen 35) is a constant and is equal to the zero-page address where the number of loops that the verb DIFAC is to execute is stored.

BUFS, BUFL and BUFH (screen 35) are constants and hold the zero-page location where temporary values are stored when DIFAC and VALAC are executed.

INCREM and INCREM*2 (screen 35) are constants and are equal to the zero-page addresses where the values of SECTION and SECTION times two are stored.

PULSECOUNT (screen 35) is a constant, the zero-page address where the number of pulses left to be collected in the present sample is stored.

SPARES (screen 35) is a constant which holds the zero-page address where the low byte of a two-byte number is stored. Together the contents of SPARES and SPARES+1 make up the two-byte value which is equal to the number of times the loop in SYNCHRONIZE is executed.

SAMPLECOUNT (screen 35) is the constant used to hold the zero-page

address where the low byte of a two byte number is stored. Together the contents of SAMPLECOUNT and SAMPLECOUNT+1 make up the two byte value which is equal to the number of samples that have been collected in the present minute.

LSP1, LSP2, HSP1 and HSP2 (screen 36) are constants, the zero-page addresses where the low bytes of four two-byte pointers are stored. The contents of LSP1, LSP+1, LSP2 and LSP2+1 point to the low bytes in RING-BUFFER that are to be used by DIFAC. Likewise, the contents HSP1, HSP1+1, HSP2 and HSP2+1 point to the high bytes used by DIFAC.

LDP, MDP and HDP (screen 36) are constants which hold the zero-page addresses where the low bytes of three two-byte pointers are stored. The contents of LDP and LDP+1 point to the low byte of the location in OUTBUFFER where the results of DIFAC are accumulated. Likewise the contents of MDP, MDP+1, HDP and HDP+1 point to the middle and high bytes of OUTBUFFER where the results of DIFAC are accumulated.

INBUF (screen 36) is a constant and is equal to the zero-page address where the page pointer to INBUFFER is stored.

LDR, HDR, LDI and HDI (screen 36) are constants and hold the zero-page addresses where the page pointers to RINGBUFFER are stored. LDR and HDR are the page pointers for the low and high bytes of the real portion of RINGBUFFER. LDI and HDI are the page pointers for the low and high bytes of the imaginary potion of RINGBUFFER.

Z (screen 36) is a constant, the zero-page address where the pointer to a one-page buffer of zeros is located.

LOR, HOR, LOI and HOI (screen 37) are constants used to hold the zero-page address where the pointers to the buffers used to figure the DC-offset are kept.

LRR, MRR, HRR, LII, MII, HII, LRI, MRI, HRI, LIR, MIR and HIR (screen 37) are constants and are equal to the zero-page addresses where the page pointers to OUTBUFFER are stored.

ADDR (screen 37) is a colon verb designed to take an address off the stack, fetch the one-byte contents of the address from memory and put the product of the contents and 100H back on the stack. ADDR is used to convert an address, where a page pointer is stored, to a two-byte address where the first memory location of that page is located.

INITLOC (screen 38) is the colon verb designed to initialize the zero-page memory locations. When INITLOC is executed, PULSES is stored in PULSECOUNT (Line 7). A one is stored in MIN#, a zero is stored in SAMPLECOUNT (Line 8) and SECTION and SECTION times two are stored in INCREM and INCREM*2 (Line 10). QUEUE2 is set to zero, QUEUE1 is set to INCREM (Line 10) and the page pointers for all the buffers are initialized (Line 13). The low bytes of LSUM and HSUM are set equal to INCREM*2 and the high bytes of HREAL and HIMAG are set equal to HDR and HDI (Lines 14-17).

INITBUF (screen 39) is a colon verb designed to erase all the buffers used in the collection program.

INITIO (screen 39) is the colon verb used to initialize the John Bell Card. When INITIO is executed, the data direction registers A and B on the first VIA are set to zero (Line 14). This initializes all bits on the ports to be inputs. The auxiliary control register on the first VIA is set to three (Line 15). This enables ports A and B on the first VIA to be latched. The peripheral control register on the first VIA is set to 40H (Line 15). This permits the interrupt flag (IFR3) to be set by a positive transition on CB2. The data direction register of port B on the second VIA is set to 81H (Line 16). This initializes port B on the second VIA

to have bits seven (PB7) and zero (PB0) act as outputs while bits six through one (PB6-PB1) act as inputs. The data register for port B on the second VIA is set to 80H (Line 16). This sets PB7 high and clears the CB1 and CB2 interrupt flags. The auxiliary control register and the peripheral control register for the second VIA are set to zero and 7FH is stored in the interrupt enable register for the second VIA (Lines 17-18), thereby resetting all interrupt flags. These actions are precautions and do not affect the operation of the program. For more information on how these commands operate see Scanlon (1980).

MACRO1 (screen 40) is used in INTERRUPT to generate the code needed to read both A/Ds once every 10 microsec. When INTERRUPT is compiled MACRO1 inserts a section of code into the code for INTERRUPT. The do-loop in MACRO1 generates a copy of the code within the loop for each height collected. These sections of code are generated continuously, each section being individually tailored to a specific height (Figure 2.2). The NOPs at the end of the loop are used to make the execution time between consecutive sets of reads equal to 10 microsec.

MACRO2 (screen 40) is used in INTERRUPT to generate the code needed to add the values stored in INBUFFER to the appropriate section of RINGBUFFER. MACRO2 uses the pointers LSUM and HSUM to indicate which section of RINGBUFFER to add the data to. The value stored in the X register tells MACRO2 whether to fetch the numbers from the real or the imaginary part of INBUFFER. MACRO2 generates one section of code for every height collected (Figure 2.2).

INTERRUPT (screens 41-42 and Figure 2.2) is the code verb used to service the interrupts generated during the collection program. The radar director puts out a 400 Hz square wave which is used to trigger

```
46E8-  8A          TXA
46E9-  48          PHA
46EA-  98          TYA
46EB-  48          PHA
46EC-  AD 00 C5    LDA   $C500
46EF-  A9 C0       LDA   #$C0
46F1-  8D 8B C5    STA   $C58B
46F4-  AD 09 42    LDA   $4209
46F7-  8D 84 C5    STA   $C584
46FA-  AD 0A 42    LDA   $420A
46FD-  8D 85 C5    STA   $C585
4700-  A9 03       LDA   #$03
4702-  8D 86 C5    STA   $C586
4705-  A0 00       LDY   #$00
4707-  8C 87 C5    STY   $C587
470A-  AE 16 42    LDX   $4216
470D-  EA          NOP
470E-  EA          NOP
470F-  EA          NOP
4710-  EA          NOP
4711-  EA          NOP
4712-  EA          NOP
4713-  CA          DEX
4714-  D0 F7       BNE   $470D
4716-  A9 80       LDA   #$80
4718-  8D 80 C5    STA   $C580
471B-  AD 00 C5    LDA   $C500
471E-  AD 01 C5    LDA   $C501
4721-  AD 00 C5    LDA   $C500   }
4724-  8D 00 72    STA   $7200   }
4727-  AD 01 C5    LDA   $C501   }  First Execution
472A-  8D 80 72    STA   $7280   }  of MACRO 1
472D-  EA          NOP           }
472E-  EA          NOP           }
472F-  EA          NOP           }
4AA5-  18          CLC
4AA6-  A5 81       LDA   $81
4AA8-  85 59       STA   $59
4AAA-  69 01       ADC   #$01
4AAC-  85 5B       STA   $5B
4AAE-  A2 00       LDX   #$00
4AB0-  A0 00       LDY   #$00
4AB2-  18          CLC
4AB3-  B1 58       LDA   ($58),Y   }
4AB5-  7D 00 72    ADC   $7200,X   }
4AB8-  91 58       STA   ($58),Y   }
4ABA-  B1 5A       LDA   ($5A),Y   }  First Execution
4ABC-  69 00       ADC   #$00      }  of MARCO 2
4ABE-  91 5A       STA   ($5A),Y   }
4AC0-  18          CLC             }
4AC1-  C8          INY             }
51C8-  A9 00       LDA   #$00
51CA-  8D 8B C5    STA   $C58B
51CD-  C6 66       DEC   $66
51CF-  D0 05       BNE   $51D6
51D1-  A9 08       LDA   #$08
51D3-  8D 0E C5    STA   $C50E
51D6-  AD 30 C0    LDA   $C030
51D9-  68          PLA
51DA-  A8          TAY
51DB-  68          PLA
51DC-  AA          TAX
51DD-  A5 45       LDA   $45
51DF-  40          RTI
```

Figure 2.2 Assembly code for INTERRUPT.

the transmitter. The square wave is also fed to the interface and connected directly to pin CB2 of the first VIA on the John Bell Card. Now if the interrupt enable register on the VIA has the CB2 interrupt enable flag set, the interrupt is passed to the Apple by the VIA bringing its IRQ line low. The 6502 CPU of the Apple then enters an interrupt service routine in the Apple monitor, ultimately vectoring through the address stored in IRQVECTORADDR. When the verb INIT executes, the address of INTERRUPT is copied into IRQVECTORADDR. This means that INTERRUPT becomes the interrupt service routine. INTERRUPT also initializes the hardware, samples the real and imaginary A/D input channels and adds the data into RINGBUFFER. A more detailed description follows here. Line numbers refer to screens 41 and 42; addresses refer to Figure 2.2.

1) The X and Y registers are transferred to the accumulator and pushed onto the processor stack (Line 5, $46E8).

2) The interrupt flag CB2 is cleared by reading DRB1 (Line 7, $46EC).

3) Timer one of the second VIA is set up to toggle PB7 every time there is a time-out. This is done by storing COH in the auxiliary control register (Line 8, $46EF).

4) DELAY1 is loaded into the timer-one clock (Lines 9-10, $46F4). This is done to allow the start-convert signal to be fine-tuned and to allow time to load the timer-one latch. The start-convert signal is produced by a one-shot with PB7 as an input. Consequently the start-convert signal can be adjusted by adjusting PB7, and PB7 can be adjusted by modifying the value of DELAY1 (Figure 2.3).

5) Timer-one latch is loaded with a three. (Lines 11-12, $4700). Every time the counter on timer-one counts to zero (timed out) the value in the latch is loaded into the counter and the counting continues. By loading a three into the latch a square wave with a 10-microsec period is generated on PB7. This is due to a five-microsec toggling period generated by a three-microsec counter with a two-microsec overhead.

6) A delay is executed to wait for the returned signal (Lines 13-15, $470A). This delay is based on an approximately 5-microsec loop that executes DELAY2 times.

7) PB7 is set high so that when the timer-one clock times out new

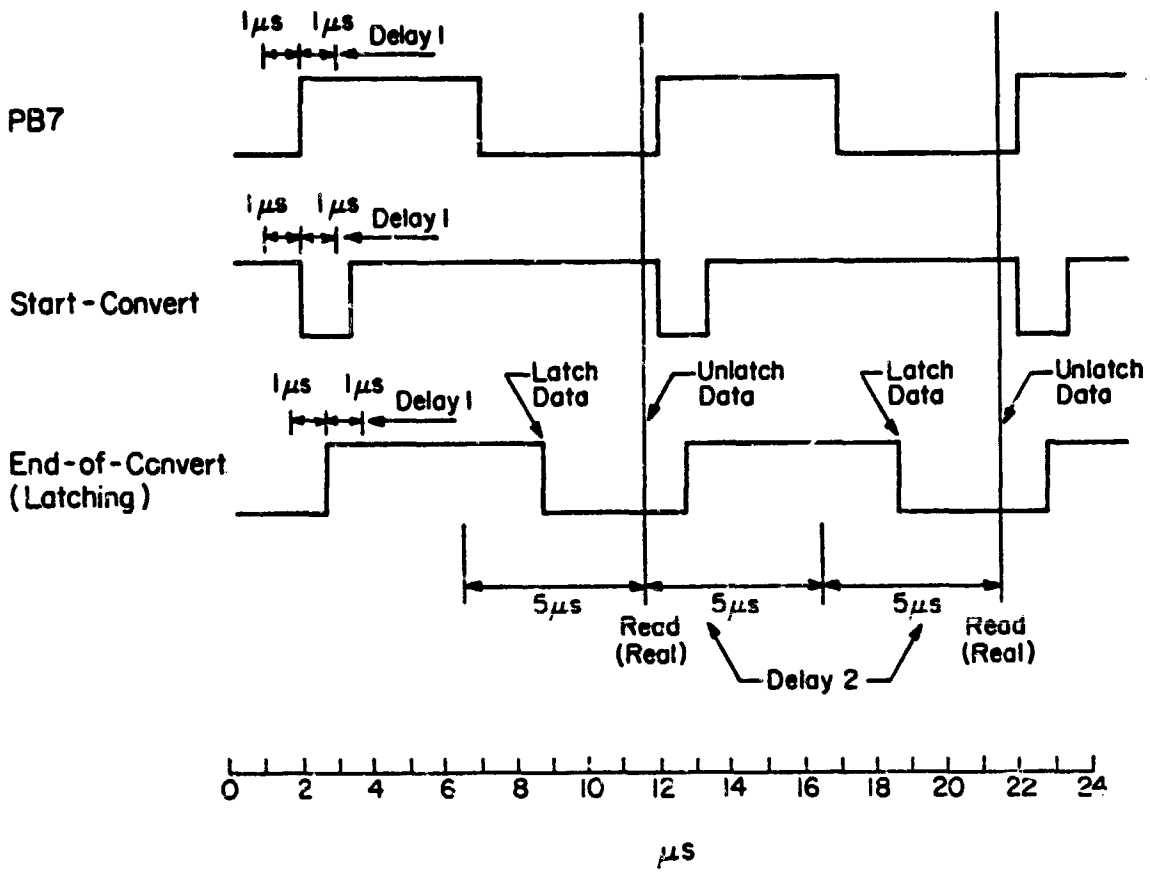

Figure 2.3 Timing diagram for data aquisition.

data are sampled (Line 16, $4716). PB7 must be high because of the timing involved between PB7 and the start-convert signals (Figure 2.3).

8) The data latched in the data registers is read and thrown away so that the new data can be read (Line 17, $4718).

9) The code generated by MACRO1 is executed (Line 18, $471E). This code is designed to leave 10 microsec between each set of reads. While MACRO1 is executing, PB7 is oscillating at 100 KHz. The 100 KHz square wave is used to produce a start-convert signal for the A/Ds (Figure 2.3). When the A/Ds have finished converting the data to digital form an end-of-convert signal is sent out. The signal is fed to CB1 and CA1 and is used to latch the new data into DRB1 and DRA1, thus the importance of having the code generated by MACRO1 precisely timed. For a detailed description of the interface see Roth (1983). A list of the connections between the interface and the John Bell Card is given here.

Dip socket A:	Pin 8	PB7 on DRB2
Dip socket B:	Pin 1-8	PB0-PB7 on DRB1
	Pin 9	CB1
	Pin 10	CB2
Dip socket C:	Pin 1-8	PB0-PB7 on DRA1
	Pin 9	CA1
	Pin 10	CA2

10) The new data is added to RINGBUFFER (Lines 5-10, $4AA6). This is accomplished by setting the pointers used by MACRO2 to the appropriate values and by executing the code generated by MACRO2. The code generated by MACRO2 is executed twice, once for the real data and once for the imaginary data.

11) The 100 KHz signal on PB7 is shut off (Line 11, $51CA).

12) PULSECOUNT is decremented and compared to zero (Line 12, $51CD). If PULSECOUNT is equal to zero it means the sample is finished and no more data is needed until the background processing is complete. If PULSECOUNT is equal to zero the interrupt on CB2 is masked so that no further additions occur to RINGBUFFER (Line 13, $51D1).

13) A "click" is produced on the internal speaker (Line 14, $51D6). When the program is running the "click" after every interrupt gives a steady 200 Hz tone as feedback to the operator that the interrupt service routine is working properly.

14) The X and Y register values are pulled from the processor stack and returned to the registers. The accumulator is also restored by reading the value at ACC (Line 15-16, $51D9).

15) The routine ends and returns control to the point where it was

interrupted (Line 17, $51DF).

SYNCHRONIZE (screen 43) is the code verb used to monitor the interrupts when the background processing is finished correlating the previous sample. SYNCHRONIZE contains a loop designed to increment SPARES every time the loop executes and to monitor PULSECOUNT to determine if enough data has been taken to complete the present sample (Lines 7-10). When PULSECOUNT equals zero SYNCHRONIZE jumps out of the loop, resets PULSECOUNT and returns to the calling verb.

MACRO3 (screen 43) is used in ADVANCE to generate the code needed to put the offset values into RINGBUFFER. When ADVANCE is compiled the code that MACRO3 produces takes the DC offset values calculated from the previous minute and stores those values into the section of RINGBUFFER that will be used to collect the next sample.

ADVANCE (screen 44) is the code verb used to update all the pointers when a new sample is started (Lines 5-9). In addition ADVANCE updates SAMPLECOUNT and determines if enough samples have been taken to complete the present minute (Lines 11-21). In order to update the pointers ADVANCE increments QUEUE1, QUEUE2, HREAL, HIMAG, LSUM and HSUM. This is accomplished by adding INCREM to each of the previous values. The pointers LSUM and HSUM are always greater than QUEUE1 by INCREM, likewise HREAL and HIAMG are always equal to QUEUE1, while QUEUE2 is always less than QUEUE1 by INCREM. These relationships make the updating of the pointers much easier. After the new pointers are in place, ADVANCE loads the DC offset into the new section. This is done with MACRO3 as explained before. Next, SAMPLECOUNT is updated (Lines 11-12) and tested against SAMPLES. If the two quantities are not equal a zero is left on the parameter stack and the program continues taking data. If SAMPLES and SAMPLECOUNT are equal

a one is left on the parameter stack and the calling verb knows that enough samples have been collected for the present minute.

TLOC (screen 45) is a constant used as a reference point when addressing the clock card.

TIM (screen 45) is a variable set up to establish a starting point in memory for an ASCII time message. TIM is initialized with D3C3H which represents "CS" the beginning of the time message. The code on Lines 7-9 inserts part of the message into the field allotted for the time message.

MACRO4 (screen 45) generates code used by MACRO5 and TIMEREAD to take the output from the clock card and put it in ASCII form.

MACRO5 (screen 45) generates code used by TIMEREAD to fill the field for the time message.

TIMEREAD (screen 46) is the code verb used to get the date and time from the Seitronics real-time clock, and is specific to that clock card. This information is inserted into the time message field set aside by TIM and the supplemental code.

AWAITMIN (screen 47) is the colon verb used to synchronize program execution to start at the beginning of each minute. AWAITMIN reads the current minute from the clock card and stores it. The verb then goes into a loop which reads the minute again and compares the result to the intial reading. When the minute changes AWAITMIN ends and returns execution to the calling verb.

CLEAROUT (screen 47) is the colon verb used to put zeros into OUTBUFFER.

VALAC (screen 48) is the code verb used to sum all the signed values of the samples for each minute. VALAC takes the section of RINGBUFFER that is indicated by the pointers LSP1 and HSP1 and adds it to the section of

OUTBUFFER that is indicated by LDP, MDP and HDP. VALAC is used by CORRELATE to calculate the DC component of the real and imaginary channels.

DIFAC (screens 49-50) is the code verb used to accumulate the absolute value of the differences needed to calculate the autocovariance function. DIFAC takes the section of RINGBUFFER that is indicated by the pointers LSP1 and HSP1 and subtracts the section of RINGBUFFER indicated by LSP2 and HSP2. The absolute value of the difference is then added to the section in OUTBUFFER indicated by LDP, MDP and HDP. The code in DIFAC is contained in a loop which automatically adjusts the input and output sections if the loop is executed more than once. The number of times the loop in DIFAC is executed is one less than the value stored in LAG.

RR, II, RI and IR (screens 51-54) are the code verbs used in conjunction with DIFAC to calculate the information needed for the autocovariance function. These verbs are designed to initialize all the pointers needed for DIFAC to get the right data to the right place.

RZ and IZ (screens 55-56) are the code verbs used in conjunction with DIFAC to accumulate the absolute values of the real and imaginary samples. RZ and IZ are very similar to RR, II, RI and IR except instead of LSP2 and HSP2 being pointed at data they are pointed at the ZEROBUFFER.

RV and IV (screens 57-58) are the code verbs used to set the pointers for VALAC. They are designed to make sure that VALAC gets the data from the right place in RINGBUFFER to the right place in OUTBUFFER.

MREGR (screen 59) is a constant, the address where the first middle byte of OUTBUFFER is stored. MREGR is used in FINDOFFSET as a reference address to where the DC-real and DC-imaginary quantities are being accumulated.

LOFFR (screen 59) is the constant used to hold the address where the

first low byte of OFFSETBUFFER is stored. LOFFR is used in FINDOFFSET as a reference address to where the offset values are stored.

OFF (screen 59) is a variable used by FINDOFFSET. OFF holds the address of the offset that is currently being calculated.

REG (screen 59) is a variable used by FINDOFFSET to hold the address where the DC quantity that is currently being used is stored.

SPLIT (screen 59) is the colon verb that takes a two-byte number off the stack and returns the low byte and then the high byte as two separate values to the stack.

FINDOFFSET (screen 60) is the colon verb used to update the offset value that is loaded into RINGBUFFER prior to the accumulation of a sample. FINDOFFSET uses two nested do-loops (Lines 4-5) to step through the heights for the real and imaginary calculations. For each calculation FINDOFFSET determines new to and from addresses (Lines 6-7) and fetches the DC values for the last minute (Lines 8-9). If the DC value is less than or equal to 7FFH, FINDOFFSET normalizes the value for one sample and subtracts the normalized value from the current offset value (Line 11). If the DC value is greater than 7FFH, FINDOFFSET adds one to the negative of the value and subtracts the negative of the normalized value from the previous offset (Lines 12-13). The new offset is then stored in OFFSETBUFFER (Lines 14-17).

DISPLACE (screen 61) is the colon verb used to relocate the middle and high bytes of OUTBUFFER. DISPLACE is used by WRITEFILE to move each odd minute of data to the end of OUTBUFFER in memory, so that two contiguous minutes of data may be written simultaneously on disk.

HEADER (screen 61) is the colon verb used by WRITEFILE to label the data disks. When HEADER calls TIMEREAD the current time is put into the

space allotted for the time message. HEADER then puts the label into the memory used for block 48H. Block 48H is then written on disk. Block 48H on the data disk contains the location used by DOS to store the disk directory. The label of a data disk can then be determined by doing a disk directory in DOS.

BLOCK# (screen 61) is the colon verb used to change the file number (used to identify each two minute data file) to the block number where FORTH will locate the data.

BMOVE (screen 62) is the colon verb used by WRITEFILE to move 4 K of memory from a designated address to four contiguous disk buffers assigned to four contiguous blocks. The first block and the beginning address are designated when BMOVE is called. BMOVE first pulls the first block number off the stack, assigns the next available disk buffer to it, and leaves the address of the disk buffer on the stack (Line 8). BMOVE then puts the file address and the file length at the beginning of the disk buffer, as required for a DOS binary file (Lines 9-11). The rest of the first disk buffer is then filled with data (Line 12). Once the first disk buffer is filled the remaining data are put into the other three buffers (Lines 13-16). Note that the last four bytes of data are lost because of the two-byte file address and the two-byte file length inserted at the beginning of the data.

ALARM (screen 63) is the colon verb used to give an audible warning when the disk is full. ALARM is made up of three nested do-loops that "click" the speaker for a length of time determined by the number on the top of the stack when ALARM is executed.

WRITEFILE (screen 64) is the colon verb used to control the output files generated from OUTBUFFER. WRITEFILE first checks to see if the

minute just collected is odd or even (Line 7). If the minute is even WRITEFILE saves the last two minutes to disk and writes the minute number just completed on the screen (Lines 8-10). If the minute is odd WRITEFILE checks to see if it is the first minute, and writes the data disk header if it is. Whether or not it was the first minute WRITEFILE executes DISPLACE to move the middle and the high bytes of the data out of OUTBUFFER and into the memory just above OUTBUFFER (Lines 11-13). Next WRITEFILE checks to see if the next to last minute was just collected. If it was WRITEFILE resets MIN# executes ALARM and writes a "CHANGE DISK" message on the screen. If any other minute besides the last one was just completed WRITEFILE increments MIN# and exits the verb.

PREVREAL and PREVIMAG (screen 65) are variable used to set up two arrays which hold the screen addresses of the symbols used in the real-time display.

PREVINIT (screen 65) is the colon verb used to initialize the display buffers. PREVINIT initializes the buffers with 0AH.

SCREENADDR (screen 65) is a colon verb used to convert the screen display line number to the screen address where the start of the line is located, in accordance with the scrambled mapping used by the Apple II. SCREENADDR takes the line number off the stack and puts the desired address back on the stack.

MACRO6 (screen 66) is used in RTDISPLAY to generate the code needed to fetch the real and imaginary samples just taken (Lines 7-8, 14-15), check to make sure the screen limits are not violated (Lines 9, 16), store the screen address of the symbols into PREVREAL and PREVIMAG (Lines 10, 17) and poke the symbols "R" (for real) and "I" (for imaginary) onto the screen (Lines 11-13, 18-20).

MACRO7 (screen 67) is used in RTDISPLAY and CLEARDISPLAY to erase the "R" and "I" symbols from the screen. The code generated by MACRO7 determines the screen addresses of the symbols from the PREVREAL and PREVIMAG arrays. The code then loads the value in the accumulator to these addresses. In order to clear the symbols the ASCII code for a space is loaded into the accumulator before the code is executed.

RTDISPLAY (screen 67) is the code verb used to display the last sample on the screen. RTDISPLAY uses MACRO7 to generate code to clear the previous display and MACRO6 to generate code to put up the new display.

CLEARDISPLAY (screen 68) is a code verb used to erase the "R" and "I" symbols from the screen. CLEARDISPLAY uses MACRO7 to generate code to erase the symbols.

ENABLE (screen 69) is a code verb used to unmask the 400 Hz interrupt signal on CB2.

DISABLE (screen 69) is a code verb used to mask the 400 Hz interrupt signal on CB2.

AMPL and AMPH (screen 69) are constants used as reference addresses in OUTBUFFER where the middle (AMPL) and high (AMPH) bytes of the absolute values of the real samples are accumulated. AMPL and AMPH are used by DISPLAYAF to display the minute averages on the screen.

RIL, RIH, IRL and IRH (screen 69) are constants used to reference the addresses where the middle and high bytes for the first lag of the RI and IR sections of OUTBUFFER are stored.

DISPLAYAF (screen 70) is the colon verb used to display the averages of the amplitude and frequency each minute. DISPLAYAF first calls CLEARDISPLAY to clear the "R" and "I" symbols left by RTDISPLAY (Line 8). The rest of DISPLAYAF is contained in a do-loop which executes once for every

height displayed. In the first half of the loop DISPLAYAF fetches the amplitude information using the reference address defined by AMPH and AMPL (Lines 9-11). This information is then divided by AMPFACTOR and adjusted for the screen (Lines 11-12). DISPLAYAF then pokes the specified ASCII character into the appropriate screen address (Line 13). The second half of the loop repeats the process given above for the frequency (Lines 14-20). Each time DISPLAYAF is executed the two ASCII characters used in the display must be specified on the stack. This feature allows DISPLAYAF to be used to erase the symbols used in the display. Erasure is accomplished by calling DISPLAYAF with the ASCII value for a "space" on the stack.

KMLABEL (screen 71) is the colon verb used to write the altitude indicators on the left of the screen. KMLABEL uses LOWHEIGHT and DISPHTS to determine the number used to label every third kilometer on the display.

INIT (screen 72) is the colon verb used to initialize IRQVECTORADDR (Line 15), the buffers, the zero page addresses and the I/O devices (Line 16). INIT initializes the display buffers, clears the screen and writes the kilometer labels on the screen (Line 17). INIT also positions the cursor at the top of the screen and defines a two line window where the messages will be displayed (Line 18).

FILLQUEUE (screen 72) is the colon verb used to fill RINGBUFFER with the initial set of samples. FILLQUEUE first sets SAMPLECOUNT to the value required to stop the data collection after enough samples have been collected to correlate the first data (Line 6). FILLQUEUE then enables the interrupt signal and begins collecting data (Line 8). When the RINGBUFFER has enough samples FILLQUEUE disables the interrupt and returns (Line 9).

CORRELATE (screen 72) is the colon verb used to correlate one minute of data. CORRELATE first enables the interrupt and begins execution of a

loop that terminates when the FINISHFLAG is set by ADVANCE at the end of the loop (Line 15-16). The loop accumulates all of the values in OUTBUFFER. The first two verbs in the loop, RV and VALAC, work together to accumulate the DC component of the real channel. Likewise, IV and VALAC accumulate the DC component of the imaginary channel. RZ, IZ and DIFAC accumulate the absolute values of the real and imaginary channels. RR, II, RI, IR and DIFAC are used to accumulate the absolute value of the differences for all of the lags. Note that when RR and II are executed with DIFAC the zeroth lag is not calculated; this is done when RZ and IZ are executed with DIFAC.

The timing in CORRELATE is very important. It is for this reason that two safeguards were built into CORRELATE. As explained in INTERRUPT, when a sample is completed the interrupt is disabled. If the correlation of the last sample is not complete the interrupt remains disabled, and data for a new sample are not taken, until the correlation processing is complete. This feature allows the program to cope with an excessive computational load by increasing the time between samples. On the other hand, if the correlation is completed before the next sample is finished, CORRELATE executes SYNCHRONIZE. SYNCHRONIZE monitors PULSECOUNT and as soon as the sample is complete SYNCHRONIZE terminates and allows the queue to advance and the next sample to be correlated.

FIXSTRAT (screen 73) is the colon verb written to amplify the lowest 20 heights in OUTBUFFER. FIXSTRAT is executed after OUTBUFFER is filled but before it is moved in memory or written on disk. FIXSTRAT multiplies the desired heights in OUTBUFFER by 16. At present FIXSTRAT is not used in the program. The reason for this is that the processing program must be modified to accommodate the amplification.

GO (screen 74) is the colon verb used to run the collection program.

GO first initializes the collection system and waits for a minute change on the clock card (Line 6). GO then collects a dummy minute of data for the initial DC calculation (Line 7). After calculating the offset (Line 8), GO begins execution of the loop that contains the collection process. GO clears the "A" and "F" symbols (Line 10), clears OUTBUFFER (Line 11) and fills RINGBUFFER (Line 12). Line 13 is used to accumulate the one minute averages in OUTBUFFER and Line 14 adjusts the offset value according to the last minute of data. Line 16 displays the "A" and "F" symbols for the last minute and Line 17 empties the disk buffers. Line 18 moves OUTBUFFER in memory or writes the previous two minutes to disk and Line 19 checks if the run is complete. GO is the main verb of the collection program. When running the program type "GO" put in a data disk and the rest will take care of itself.

7.3 Modifications

The collection program described in this chapter was put into use August 1984. Previous to that time a version of the program was used which did not require the use of the Number Nine Booster Card. The previous program collected 20 heights and alternately sampled the real and imaginary channels every other pulse. The alternate sampling was necessary because of the minimum load and store time of 8 microsec. To load and store both channels require 16 microsec when only 10 microsec was available. The use of the booster card allowed the load and store to be done in under 10 microsec.

In order to implement the use of the booster card several screens needed to be changed, and carry the header "ADR 8/84". Other changes such as the number of heights collected and data disk labeling have been included in the latest version of the program. A copy of the unmodified

screens is given in Appendix B. Those screens not shown in Appendix B may be found in Appendix A.

3. COHERENT-SCATTER ANALYSIS PROGRAM

3.1 Previous Program

The objectives of the coherent-scatter analysis program are to read the two-minute binary files stored on disk by the collection program, calculate the power velocity and correlation time for each minute, according to the equations of Section 1.2, and write that information to three text files on a separate disk.

Previously the analysis program was broken into two separate programs. The first program is called ANAL4 and is shown in Appendix C. After some initialization (Lines 1-60) ANAL4 requests that the user type in the number of files to be processed and the name of the binary intermediate file to be used as an output. ANAL4 stores these values in FI and A$ respectively (Lines 98-99). The next task for ANAL4 is to read the first data file into a fixed memory location (Line 104). The format of the data file is explained in Chapter 2 and is briefly reviewed here.

Each file has two minutes of data stored in it. The first minute is stored in the second half of the file, while the second minute is stored in the first half. Each file is 4 K bytes long with 2 K dedicated to each minute. Each minute of data has four two-dimensional arrays RR, II, RI, IR as described in Section 2.1. The format for one minute of data is shown in Table 3.1. The 32 bytes on each line pertain to the 32 possible heights that can be used with an eight-section system. When the collection program writes the data file to disk the last four bytes of the file are lost. This is due to the fact that when a binary file is written to disk a two-byte load address and a two-byte file length are written along with the file. These four bytes are stored as the first four bytes of the binary file.

Table 3.1 Format for binary data files.

BYTES	QUANTITY	BYTE	SECTION QUANTITY
0-31	RR	MEDIUM	NOT USED
32-63	RR	MEDIUM	LAG 0
64-95	RR	MEDIUM	LAG 1
:	:	:	:
:	:	:	:
224-255	RR	MEDIUM	LAG 6
256-287	II	MEDIUM	NOT USED
:	:	:	:
:	:	:	:
480-511	II	MEDIUM	LAG 6
512-543	RI	MEDIUM	NOT USED
:	:	:	:
:	:	:	:
736-767	RI	MEDIUM	LAG 6
768-799	IR	MEDIUM	NOT USED
:	:	:	:
:	:	:	:
992-1023	IR	MEDIUM	LAG 6
1024-1055	RR	HIGH	NOT USED
:	:	:	:
:	:	:	:
1248-1279	RR	HIGH	LAG 6
1280-1311	II	HIGH	NOT USED
:	:	:	:
:	:	:	:
1504-1535	II	HIGH	LAG 6
1536-1567	RI	HIGH	NOT USED
:	:	:	:
:	:	:	:
1760-1791	RI	HIGH	LAG 6
1792-1823	IR	HIGH	NOT USED
:	:	:	:
:	:	:	:
2016-2047	IR	HIGH	LAG 6

Thus when the file is written to disk the last four bytes of the binary file are truncated, namely, the high byte of the highest lag for the highest four heights. This becomes important if all 32 heights and all six lags are being used. Since ANAL4 only uses 20 heights and two lags, the loss of the last four bytes does not present a problem.

After reading the first data file ANAL4 stores the first minute of data into an array called A(I,J,K) (Lines 133-150). In order to calculate the correlation and phase for the first heights, the magnitude G(I) and phase V(I) of the autocovariance function are calculated for the first and second lags (Lines 187-260). After the power X1 is calculated (Line 265) a check is made to see if enough power is present to calculate the velocity and correlation time. If there is not enough power ANAL4 sets the velocity X3 and the correlation time X2 to a default value (128 for ANAL4) and continues with the next height (Line 272). If there is enough power ANAL4 checks the shape of the autocovariance curve. If the magnitude of the autocovariance is greater at the second lag than at the first lag, or if the magnitude of the autocovariance of the second lag is less than one, ANAL4 sets the correlation time X2 to a default value (128) and sets the velocity X3 equal to that calculated for the first lag (Line 273). Provided that there is enough power and that the magnitude of the autocovariance is within the limits, ANAL4 calculates the correlation time X2 (Line 274).

ANAL4 was designed to produce an output of power, velocity and correlation time in one-byte integer form, so it uses a scale factor. For instance, the velocity is calculated at one-tenth its final value. This allows ANAL4 to store a wider range of velocities in an eight-bit number. After all the criteria have been checked and the three quantities have been

put in eight-bit form, ANAL4 stores the three values into memory (Line 285). This process is repeated for each height and then for each minute until all the data on side A of the collection disk (first hour) is complete. If less than one hour of data was collected ANAL4 fills the remaining memory space with the default value 128 (Line 290-295). The last task for ANAL4 is to save all the processed data (in memory) to an intermediate binary file on disk (Line 310). ANAL4 is then run on side B (second hour) of the collection disk. In order to put the two hours of data together, both the binary files are loaded into memory in consecutive address locations. Then one intermediate binary file of double length is saved back to disk.

The second program CONVERT84.1 takes the intermediate binary file and outputs three serial ASCII text files. These files are in a format compatible with those previously generated by the PDP-15 computer (Roth 1982) and represent power (POWW), velocity (VELL) and correlation time (CORR). CONVERT84.1 calculates the minimum, maximum and mean of each of the three quantities. Finally the serial text files are written. The first line of the text file contains the title and the date when the data was collected. The second and third lines show the hour and minute of the start time. The fourth line contains the number of records (minutes) contained in the file. The fifth, sixth and seventh lines have the minimum, maximum and mean values of the numbers in the file. The eighth line has the base-height used in the collection of the data. After writing these quantities to disk CONVERT84.1 then writes the data that is stored in the intermediate binary files to disk. Before the values are written to disk they are returned to their original form. All numbers that were originally negative are restored by subtracting 256 and each value is multiplied by the scale factor used in ANAL4. The scale factors are ten for the velocity and two for the

power and correlation time. The numbers are written on disk in the format shown in Table 3.2. Note N is the number of minutes collected.

The final form of the three quantities, power, velocity and correlation time, have units of centibels centimeters-per-second and centiseconds respectively. This selection of units allows the values to be stored in the text files as integers. To recover the data in bels, meters-per second and seconds, the integer is divided by 100.

CONVERT84.1 ends when all the files have been completed.

3.2 System Limitations

The processing programs ANAL4 and CONVERT84.1 have four inherent limitations which are described below.

(1) ANAL4 and CONVERT84.1 only allow for a maximum of 256 different values for any of the calculated quantities. This is due to the fact that only eight bits are used to represent each number. In order to accommodate this, the maximum range of values to be considered severely limits the resolution of each number. For instance if the extreme velocities to be considered were $\pm$12 m/s, the resolution at best could only be .094 m/s. ANAL4 and CONVERT84.1 use a scale factor of ten and calculate the velocities in decimeters-per-second. This gives a resolution of 0.1 m/s and a maximum range of $\pm$12.7 m/s. Since the existing plotting routines do not plot zeroes, all velocities in the range of $\pm$0.05 m/s are not plotted and look like missing data. Similar arguments can be made for power and correlation time data.

(2) ANAL4 and CONVERT84.1 are complicated to use. The processing of two binary files (one for each time ANAL4 must be run) as well as the retrieval and joining of these files leaves considerable room for user error. Memory locations and file lengths must be kept straight in order to get

Table 3.2 Format for text data files.

HEIGHT	MINUTE
1	1
1	2
1	3
:	:
:	:
1	N
2	1
2	2
:	:
:	:
2	N
:	:
:	:
:	:
20	1
20	2
:	:
:	:
20	N

the correct information to CONVERT84.1.

(3) Because of the improvements to the collection program (Chapter 2) it has become necessary that ANAL4 and CONVERT84.1 have the ability to process up to 60 heights of data for each disk. This change increases the usage of ANAL4 and CONVERT84.1 by a factor of three. This increase in the volume of data multiplies the difficulties in using the processing programs.

(4) Since the range of heights now includes altitudes from the stratosphere, the maximum values are no longer as meaningful. This tends to make the decreased resolution even more unbearable. For instance, typical velocities in the stratosphere are $\pm$1.5 m/s. With a 0.1 m/s resolution only 30 different velocities can be displayed.

3.3 Modifications

In order to improve the processing programs and to allow the processing of data from the improved collection program several modifications have been made.

(1) To allow a greater range of values the number of bits used to store one piece of data has been changed from 8 to 16. Using the previous example, the range can now be $\pm$327 m/s. Since the range now more than adequately covers the possible values the scale factor of ten is no longer necessary. The resolution of the values now is 0.01 m/s. Because the increased range is much larger than needed, a scale factor of one-tenth or smaller is now possible. This would allow an accuracy of 0.001 m/s or better. Clearly the accuracy of the output data is no longer limited by the processing system but is limited by the collection system. This modification has led to an increased number of valid data points and a much better plot.

(2) In order to streamline the processing programs several modifica-

tions have been made. ANAL4 and CONVERT84.1 have been merged into one program called PROCESS. PROCESS has a number of advantages over ANAL4 and CONVERT84.1. PROCESS is much faster to use. The simple mechanics of only having one program decreases the processing time. PROCESS does not use intermediate binary files. This leads to less disk usage and less confusion. PROCESS can do up to two hours of data analysis in one run. It is no longer necessary to glue files together.

(3) By adding two variable DISP and SECTION, PROCESS has the ability to analyze data collected with the new collection program. SECTIONSIZE controls the input data file format while DISP allows PROCESS to analyze data from three different height ranges (LOW, MEDIUM, HIGH).

(4) The increased number of bits used to represent each value also improves the quality of the data seen at altitudes in the stratosphere. The number of possible velocities in the stratosphere has increased from 30 to 300. This change causes the structure of the data to be greatly enhanced.

3.4 The New Analysis Program

The new analysis program is called PROCESS and is shown in Appendix E. Some points of interest are discussed here. HIMEM, INFILEADDR and OUTFILEADDR are used to configure the available memory space. HIMEM sets the address for the highest memory location available to PROCESS and its variables. INFILEADDR is the address where the input-file will be loaded into memory. To accommodate the size of the input-file this constant must be at least 4096 less than the highest available memory location. OUTFILEADDR is a memory location where the data will be stored before being written to the text file. Since 14,400 bytes are required for a two hour 16-bit file, OUTFILEADDR must be at least 14,400 less than INFILLEADDR. SECTIONSIZE should be set equal to the value of SECTIONSIZE in the collec-

tion program. THRESHOLD is used to modify the acceptable power level. THRESHOLD is a variable and is used to replace the constant 20 used in ANAL4. REFILE is a variable used to load the data file into memory. REFILE replaces H in ANAL4. K1, K2 and K3 are constants used in calculating velocity, correlation time and power. Statements containing "PRINT D$..." are used to execute DOS commands from a BASIC program. The colons in front of several of the statements are used to indicate the nested loop level and have no effect on the statements they precede.

A brief description of PROCESS is given here. PROCESS first initializes all integers and arrays (Lines 10-20). Next, PROCESS determines the height range and initializes some constants (Lines 30-170). PROCESS then fills the input array and calculates the power, velocity and correlation time for the first minute. There are two functional differences between ANAL4 and this part of PROCESS.

(1) The data is now compressed to 16 bits rather than 8 bits. So, all values are clipped to $\pm$32,676 and put into a positive two-byte form by adding 65,536 to all values less than zero.

(2) The criterion for the calculation of the correlation time has been changed. PROCESS checks to see if the magnitude of the covariance function at the first lag is five times greater than the magnitude of the covariance function at the second lag. PROCESS also checks to see if the magnitude of the covariance function is greater at the second lag than at the first. If either condition is true PROCESS sets the correlation time to a default value and assigns VEL equal to the velocity calculated at the first lag (Line 2510).

After all the data has been analyzed and stored in memory PROCESS immediately begins creating the text files. PROCESS collects the header in-

formation (Lines 4330-4410), calculates the minimum, maximum and means (Lines 4420-4670), and writes the three files to disk (Lines 4073-4315).

PROCESS is written in several parts and is easily followed. After some initializing PROCESS asks for the height range to be analyzed. Next, PROCESS asks for the number of files to be analyzed and a pause is put in to remind the user to put in side A of the data disk. PROCESS then reads and does the analysis on all the side A files in much the same manner as ANAL4. When side A is done PROCESS asks the user to put in side B of the data disk. PROCESS then does the analysis on side B and puts the output data right next to the data for side A. PROCESS then asks for the "TEXT DISK". The TEXT DISK is simply a separate disk which will be used to permanently store the output text files. Next, PROCESS asks for some header information for the text file. It then calculates the minimum, maximum and mean for all the data and writes these along with the data to the text files. When this is done the text file is complete and PROCESS restarts itself.

4. COHERENT-SCATTER DATA ON MAGNETIC TAPE

4.1 Introduction

In order to establish a forum for data exchanges and discussions, the MAP (Middle Atmosphere Program) of SCOSTEP (Scientific Committee On Solar Terrestrial Physics) has set up a project called MSTRAC (MST Radar Coordination), under the chairmanship of P. K. Rastogi of Case Western Reserve University. In accordance with MSTRAC, a data exchange of MST radar data has been initiated. Several countries (U.S.S.R., Czechoslovakia, Federal Republic of Germany and China) have indicated an interest in using MST radar data to aid their studies of the middle atmosphere.

In order to share coherent-scatter data with other users it is necessary to transfer the data to a different medium. Potential users of the data have requested that the transfer be made to 1600 bpi, 9-track, IBM readable magnetic tape. The procedure for the transfer of data from floppy disk to magnetic tape is described in this chapter. At this point some preliminary discussion is in order.

(1) Magnetic tape 3600 ft long can be purchased from CSO on campus. If a different length is desired the tape must be purchased elsewhere.

(2) In order to use a magnetic tape the tape must be checked into the tape room at CSO. When tapes are checked in, a "tape name" and a "rack designation" are assigned. The "tape name" is one to six characters in length and is chosen by the user. The "rack designation" is four characters long and will be assigned. The "rack designation" is either temporary or permanent. All tapes to be left less than 30 days are temporary and have a "rack designation" of TEMP. All other tapes have a different desig-designation assigned by the operator. To retrieve a tape simply go to the

CSO tape room and tell the operator the "tape name" and the "rack designation".

(3) There are two ways to submit jobs to the Cyber. Jobs can be submitted interactively or in a batch mode. To execute jobs interactively simply type in the first command and wait for the prompt. Then type in the second command, so on and so forth. Interactive is the method used to transfer floppy disk files to Cyber permanent storage. When reading, writing or copying tapes it is recommended that the batch mode be used. To use the batch mode, create a text file on the Cyber with the following format.

```
/JOB
/NOSEQ
WIZARD.
#########                          (ID NUMBER)
XXXXXX.                            (PASSWORD)
BILL,XXXX,PS####.                  (CHARGE NUMBER)
PRINT/RJE=EE.
(NORMAL CYBER COMMANDS)
    :          :
    :          :
    :          :
```

To execute the batch job type "SENDJOB,filename." return. The Cyber will then put the job on an execution queue. The job will now execute on its own. The users can go on to something else, or log-off. To determine if the job is completed type "QUERY" return. Cyber will respond with "NO JOBS QUEUED" if the job is done. Note: the print statement in the header of the batch file causes all output generated by the batch job to be routed to EEB computer room. However, a program listing may still go to the CSO computer room. Due to the fact that the ID number, password and charge number are on the program listing, it is important to retrieve all listings from both locations.

(4) The procedure described in this chapter was used to create a mag-

netic tape with data collected during April 1978. The master tape is called APRILM and is located on rack G454. A copy of the master tape called APRIL1 is resident in the Aeronomy Laboratory. Due to the amount of data present approximately 300 ft of tape are used. Therefore all copies made for distribution have only 300 ft of tape on them. These tapes are named APRIL2 through APRILB.

4.2 Format of Tape Files

In this section the format used to store coherent-scatter radar data on magnetic tape is described. This particular format was chosen because it is simple to follow and easy to use. The data files stored on magnetic tape use a format involving a large number of small files. Each file represents a single quantity (power, velocity) for a two-hour period. The ability to easily access each two-hour period of data from individual files makes this format preferable to one with only a very few large files.

The first file written to the magnetic tape is a "header file". The header file for April 1978 is shown in Appendix F. The header file describes in detail the conditions under which the data was collected, the format of the data files on the tape and a menu of the files on the tape. The format of the data file was chosen because of its relative simplicity and its economical use of the magnetic tape. The format of the tape data files is very similar to that of the floppy disk data files (Section 3.1). The only change occurs in the record length. In serial text files the record size is variable so as to fit each individual number. The coherent-scatter analysis program writes serial text files with one number per record. Tape files have a fixed record length which must be large enough to accommodate the largest possible number. It was determined that one value could at most use six characters. Six characters allow a five digit

number and a sign indicator. To allow economical use of the magnetic tape an 80 character record is used. This record length allows 11 different values to be stored in one record. Each record has six characters per number plus one blank space between each value. Three characters are left blank at the end of each record.

4.3 Reformatting Disk Text Files

As explained in Section 4.2 it is necessary to reformat the data files into 80 character records. There are two possible solutions to the task of reformatting the text files. The files could be reformatted, stored back on floppy disk and then transferred, or the files could be reformatted once the transfer has taken place. Because of ease and the accessibility of the Apple II microcomputer, the former of the two solutions was chosen. A small BASIC program was written to read the serial text file, reconfigure the data into 80 character records and write a new text file with the new format. The BASIC program is shown in Appendix G.

4.4 Transfer of Text Files to Cyber

Before the data files can be stored on magnetic tape they must be transferred to the Cyber computer. The procedure for executing the transfer is described below.

Once the text files have been reformatted they can be transferred via a modem link to the Cyber computer. The transfer program used is a public domain terminal program available from CSO called Apple Term version 3.0. The procedure is as follows. Boot up the Apple Term disk. When the Apple asks for a phone number, type in the number of the Cyber (Nosa or Nosb). Once the link has been established type a blank carriage return. Now sign-on in the normal procedure. After sign-on type "ICE,filename" return. When the Cyber responds with a "??", type "I" enter. The Cyber is now

ready to accept data from the terminal. Next type escape "T". From here the Apple Term program will take over and ask for the filename of the file to be transferred and the prompt used. Make sure that the disk with the text file present is in the disk drive. Now type in the filename return for the filename and a "?" return for the prompt. The "?" tells the Apple Term program to wait for a "?" before transferring a new line. After typing in the prompt, no more user inputs are needed until the file transfer is complete. Upon completion of the transfer, the Apple will beep. Type in a blank carriage return. This will terminate the file. A "??" prompt will then appear. Type in "ER" return. This exits the editor and saves the new file in permanent storage.

4.5 Reading and Writing Magnetic Tapes

There are many different ways to write information to a magnetic tape. The procedure outlined below describes one way for a Cyber computer to write a magnetic tape that an IBM computer can read.

To write a file to tape first send a lable statement to the Cyber.

LABEL(TAPE,VSN=tapename-rack designation,PO=W,F=S,LB=KU,CV=EB)

This label statement tells the computer operator to mount a tape called tapename. The tape may be found at the location rack designation. "TAPE" is a variable used to refer to all operations on the tape. "PO" can either be set to "R"ead or "W"rite. "F" is set equal to "S" to indicate that the format on the tape is "strange". "LB" is set equal to "KU" and indicates the tape is unlabeled. "CV" is set equal to "EB"; this will cause data written to the tape to be stored in the Ebcdic code.

Once the label statement has been executed a request for the TBLOCK routine should be made.

GRAB,TBLOCK.

This command will fetch TBLOCK into local memory.

Now, the first file to be written to memory should be fetched from permanent storage.

```
GET,dfilel.
```

After dfilel is in local memory the file can be written to tape.

```
TBLOCK(DISK=dfilel,RECSIZE=80,BF=50)
```

This command tells Cyber to write dfilel to tape using a record size of 80 characters and a buffer size of 50 records. One buffer represents a block of data stored on tape. If another file is to be written to tape simply fetch it to local memory and execute another TBLOCK. This is possible because the Cyber just finished the first file. However, the tape might not always be in the correct position. To reposition a tape at the beginning of file, simply rewind the tape and skip over the appropriate number of files.

```
REWIND,TAPE.
SKIPF,TAPE,n,C.
```

Here n is the number of files to skip over.

When the session is over it is important to return the tape to its rack.

```
RETURN,TAPE.
```

This tells the computer operator to put the tape back, it also will free up a tape drive for another user.

In order to verify the data it might be necessary to read the data back off the tape. To read a tape issue a label statement with "PO" set equal to "R". Then position the tape to the beginning of the desired file. Next, fetch DEBLOCK and execute the DEBLOCK command.

```
GRAB,DEBLOCK
DEBLOCK(DISK=file,RECSIZE=80,BF=50)
```

The parameter "file" indicates the name of the local file where the data read from tape will be put.

4.6 Making Copies

After the master tape is complete it may be necessary to create several copies of the tape for distribution. The procedure for making copies is outlined below.

In order to make a copy of a tape two tapes must be checked in at the tape room. To allow for a lengthy execution time, the time limit for the user number must be increased to 20 time units.

```
SETTL,20.
```

Next, a resource allowance of two tape drives must be requested.

```
RESOURC,PE=2.
```

Then, two label statements must be issued.

```
LABEL,TU,VSN=newtape-rack,F=F,FC=6000,LB=KU,PO=R.
LABEL,FRM,VSN=oldtape-rack,F=F,FC=6000,LB=KU,PO=W.
```

After the label statements have been issued a copy command must be executed.

```
COPY,FRM,TU,TC=EOD,V=YES.
```

This will put what is on tape FRM on to tape TU. When this command is executed a verify will also be implemented. The Cyber will respond with "VERIFY GOOD" if a good copy was made. Once the copy is complete return both tapes.

```
RETURN,TU.
RETURN,FRM.
```

5. DISCUSSIONS AND SUGGESTIONS FOR FUTURE WORK

The new collection program has many advantages over the old one. The increased speed of the booster card provides improved timing when servicing interrupts. When an interrupt is detected there is an uncertainty as to when the interrupt service routine starts execution. This is because the processor must finish the current instruction before servicing an interrupt. Because the booster card reduces the average execution time by approximately a factor of 3.5 the uncertainty time is also reduced by a factor of approximately 3.5.

Because of the ability to sample both the real and the imaginary channels after each pulse, the collection program now collects twice as much data per height. The increase of data has lead to a better signal-to-noise ratio. The new collection program collects up to 60 heights, three times the number of heights collected by the old program. These 60 heights need not be contiguous. For instance, it is possible to collect data from 9-39 km and 49-109 km. The increase in data and heights has not lead to an increase in the storage space needed by the new collection program.

Analysis of the data from sample waveforms fed directly to the interface shows that the collection program is working very well. Comparisons between data collected with the PDP-15 and the Apple support the test results. A combination of improvements to the collection program and the analysis program has increased the quality of the velocity and correlation time plots. Sample plots generated with the new programs are shown in Figures 5.1-5.6.

In the lower altitudes the correlation time of the data is significantly greater than in the higher altitudes. Consequently, the values cal-

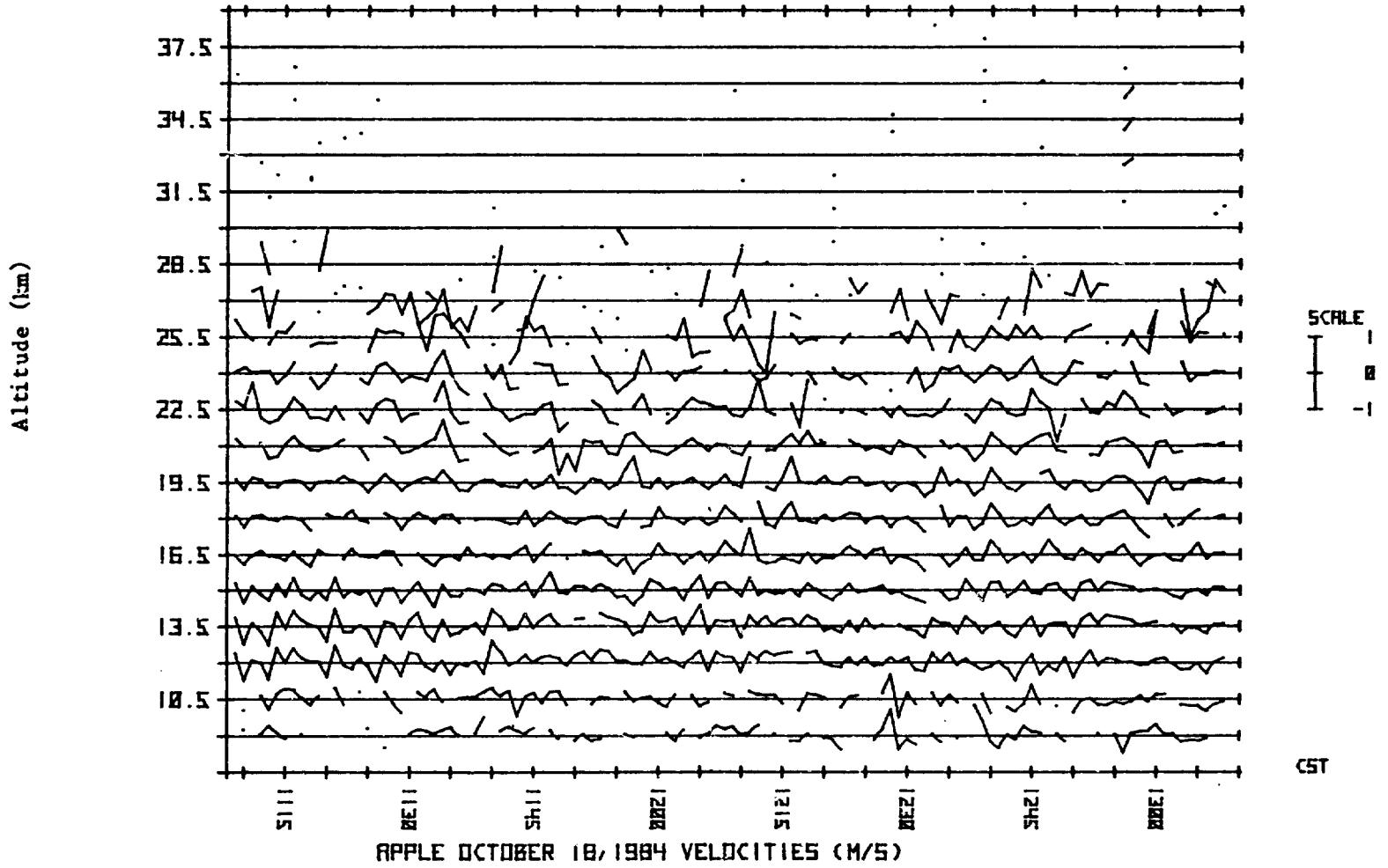

Figure 5.1 Line-of-sight velocity at Urbana beginning at 1109 CST on October 18, 1984.

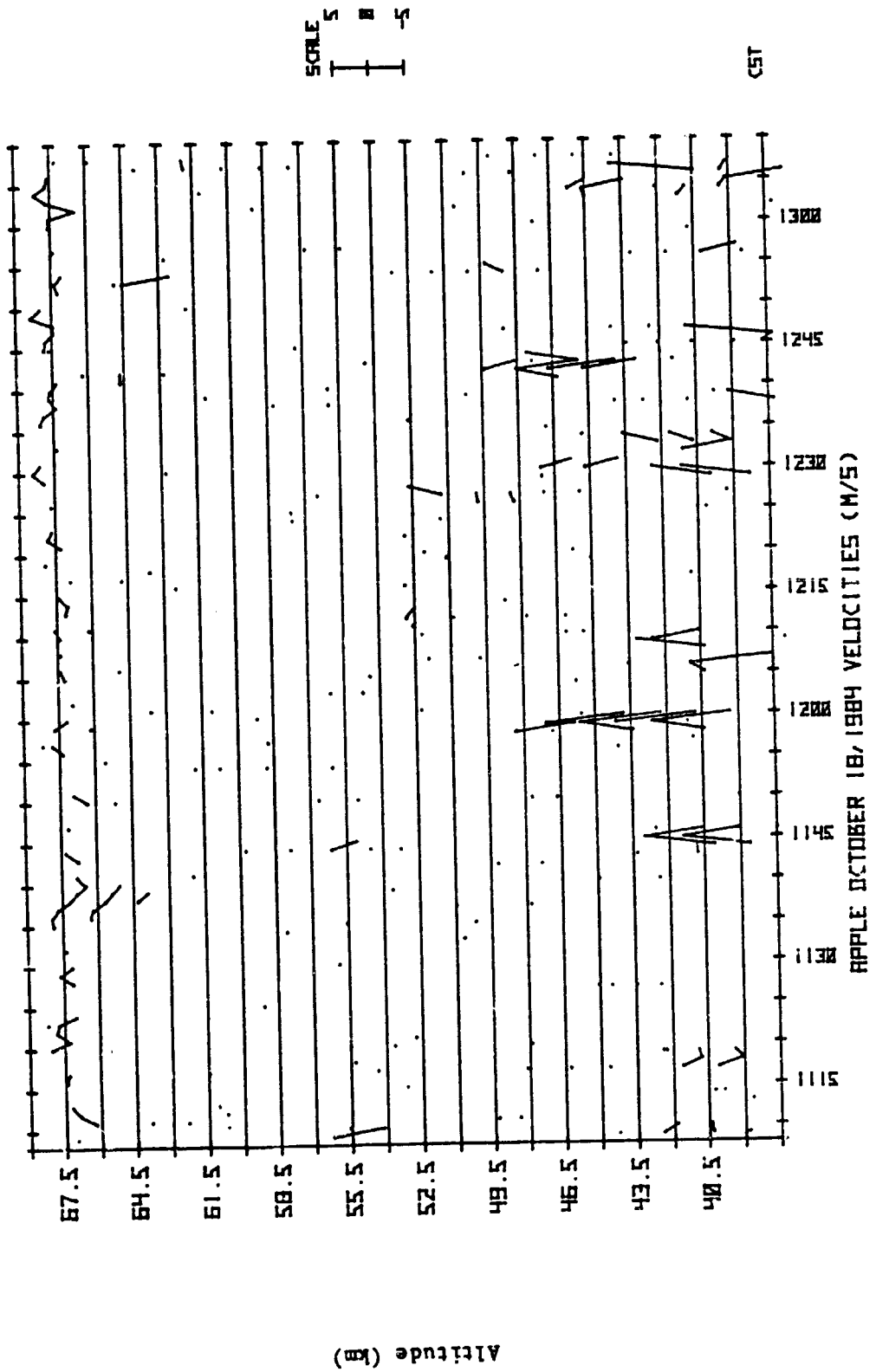

Figure 5.2 Line-of-sight velocity at Urbana beginning at 1109 CST on October 18, 1984.

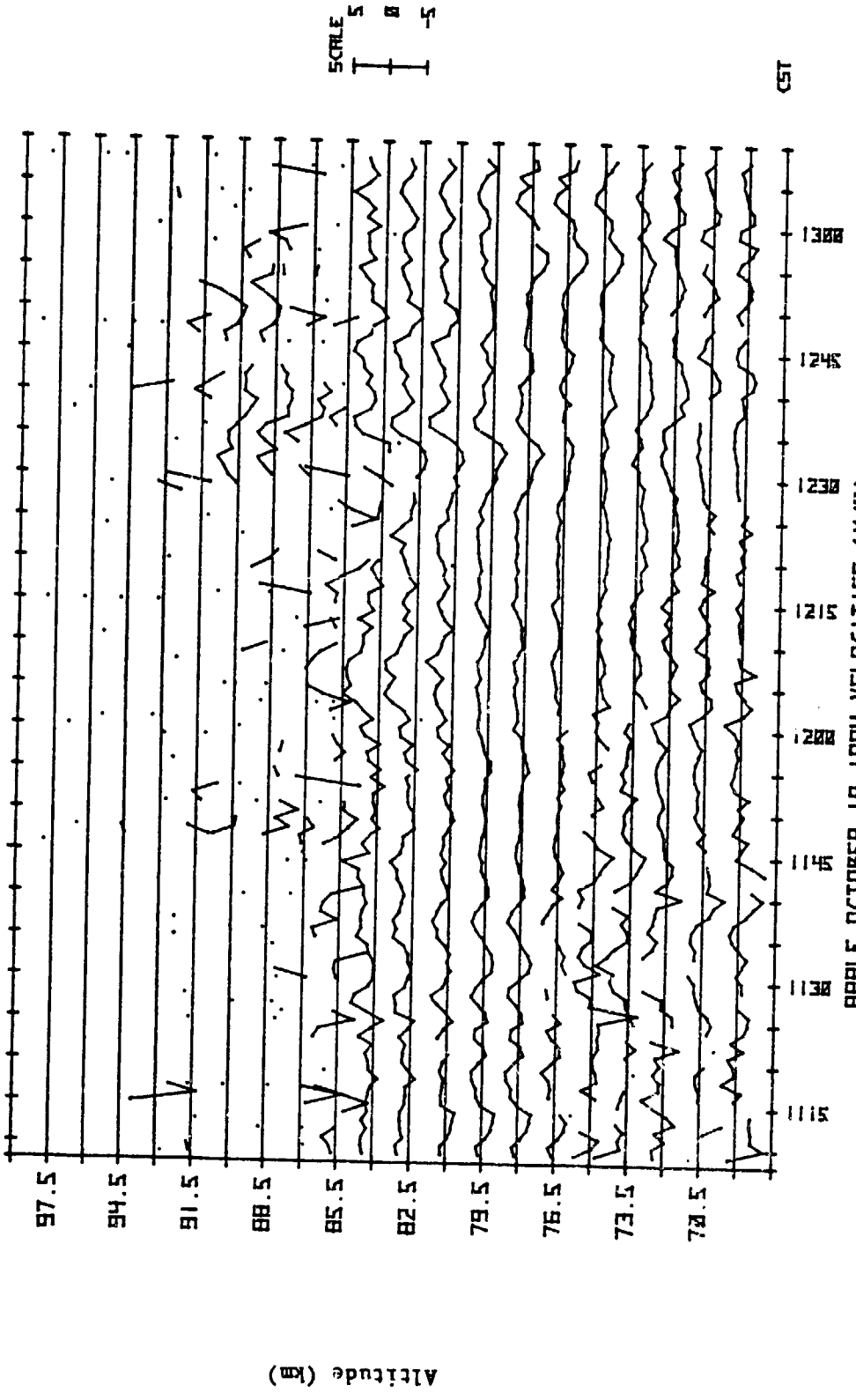

Figure 5.3 Line-of-sight velocity at Urbana beginning at 1109 CST on October 18, 1984.

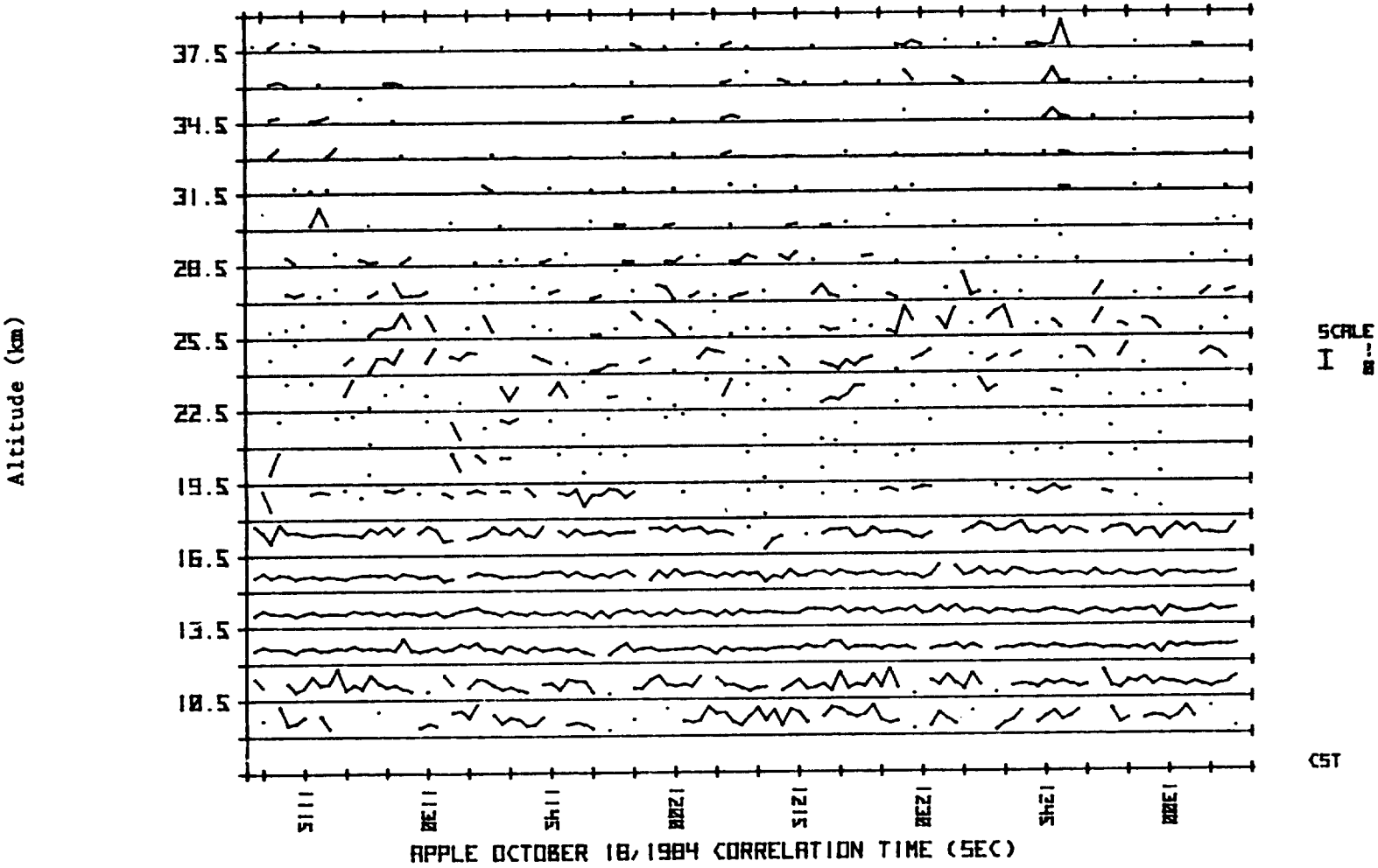

Figure 5.4 Correlation time at Urbana beginning at 1109 CST on October 18, 1984.

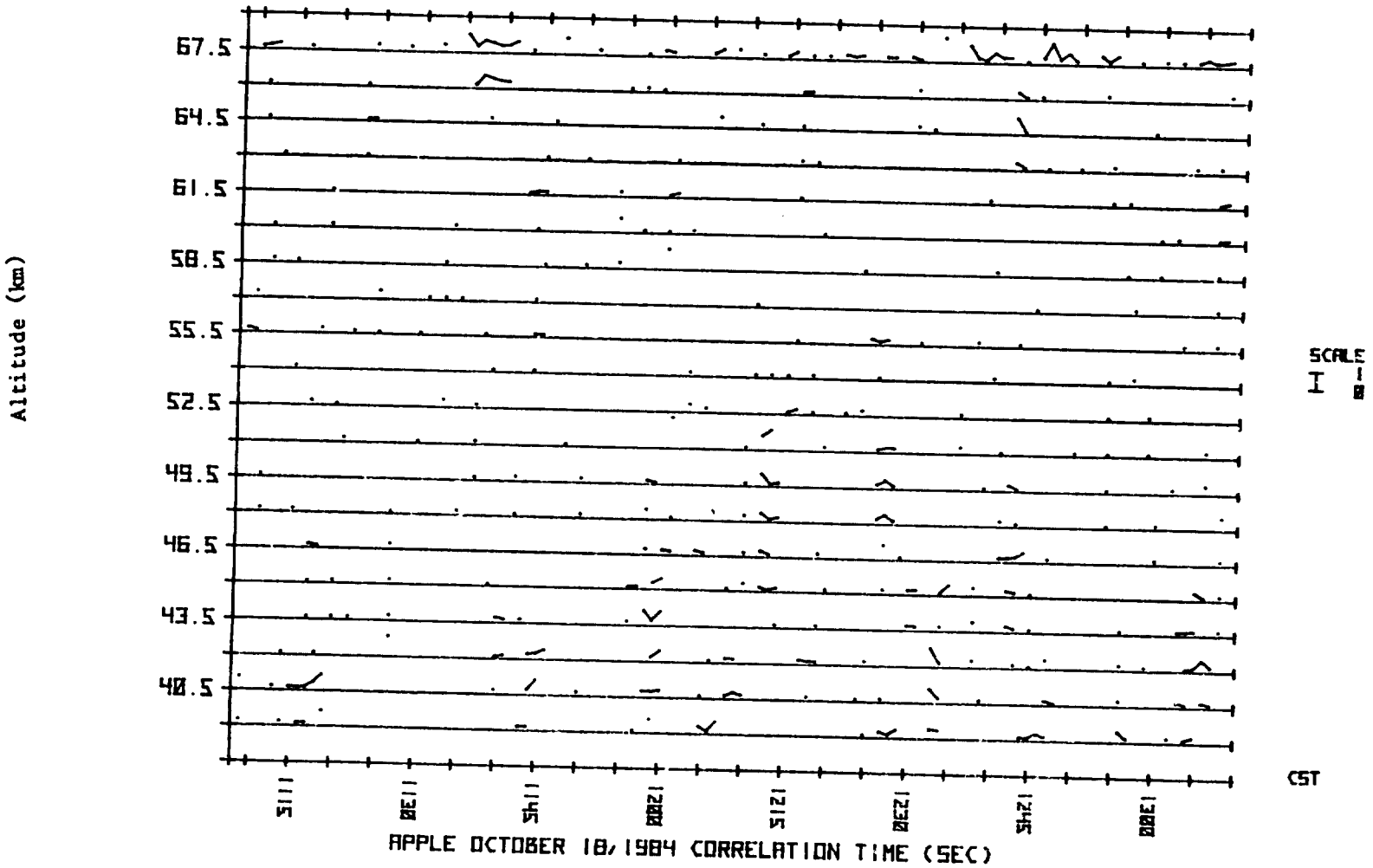

Figure 5.5 Correlation time at Urbana beginning at 1109 CST on October 18, 1984.

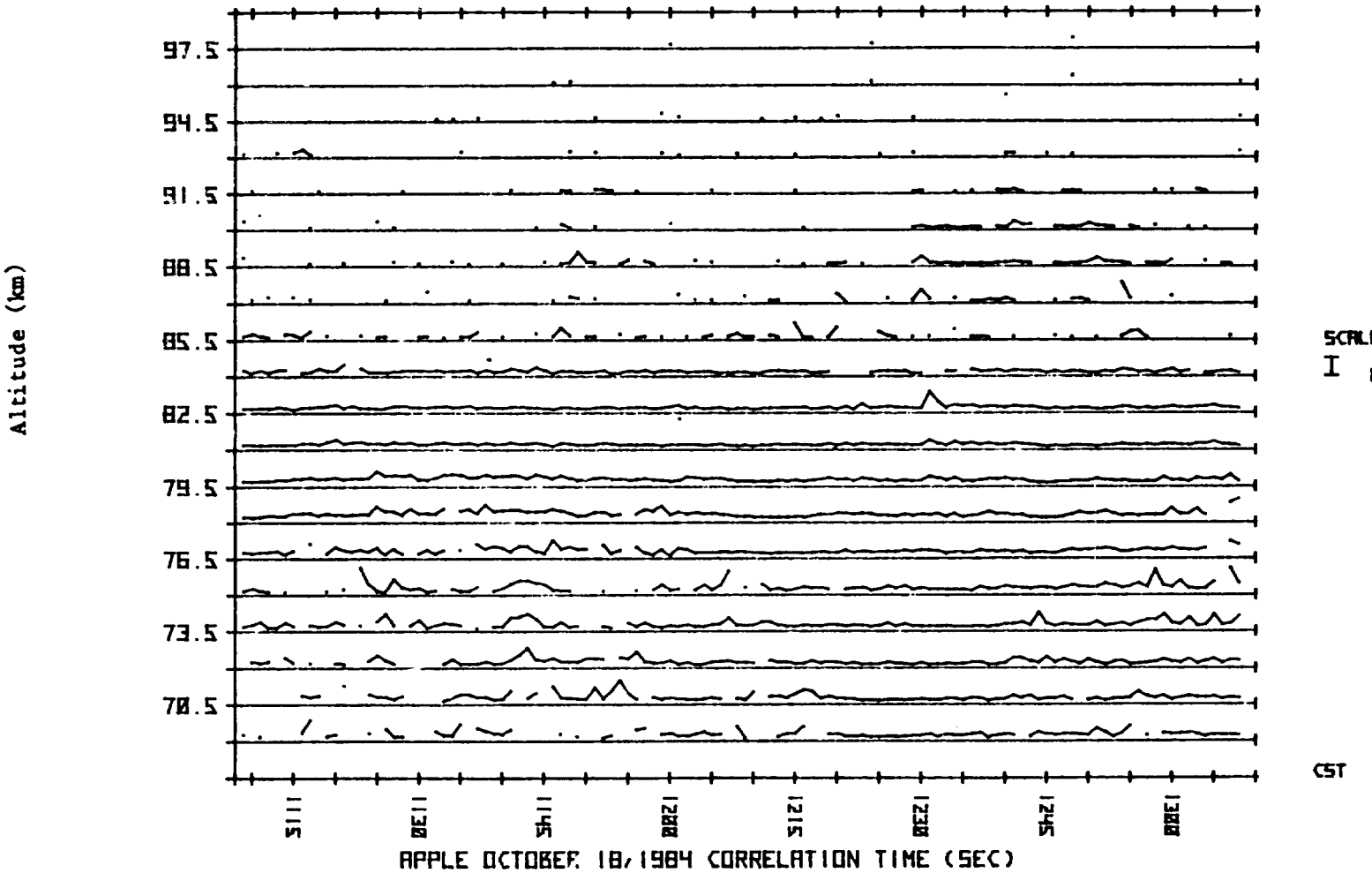

Figure 5.6 Correlation time at Urbana beginning at 1109 CST on October 18, 1984.

culated by DIFAC are much smaller. To amplify these values the FIXSTRAT verb was written for the collection program. However, this verb cannot be used without loss of information in the power plots. Small modifications in the analysis program can be implemented to allow the use of FIXSTRAT.

Reducing the overshoot of the transmitted pulse would also improve the quality of the data at the lowest altitudes. In addition if the overshoot were reduced the collection program could be modified to collect data from lower altitudes. These changes could be used to help study the correlation between winds in the upper, middle and lower atmosphere. Upgrades to the Urbana system allowing a decreased overshoot are currently being implemented.

APPENDIX A
Listing of the Collection Program

```
SCR # 30
 0 ( RADAR/1                ADR 8/84 )
 1
 2 (
 3 COHERENT-SCATTER-RADAR DATA-ACQUISITION
 4 PROGRAM WRITTEN OCTOBER 1981 BY SIDNEY
 5 BOWHILL, REVISED JUNE 1983. REVISED
 6 AUGUST 1984 BY ANTHONY RENNIER. RESEARCH
 7 SUPPORTED BY NATIONAL SCIENCE FOUNDATION
 8 AND BY NATIONAL AERONAUTICS AND SPACE
 9 ADMINISTRATION.
10
11 ASSISTANCE OF FRANCIS KEASLER AND DAVID
12 PADGITT IS ACKNOWLEDGED. ORIGINAL
13 FIG-FORTH SYSTEM BY FORTH INTEREST
14 GROUP, SAN CARLOS, CALIFORNIA.
15
16 ASSEMBLER, EDITOR AND METACOMPILER BY
17 GEORGE LYONS, JERSEY CITY NJ.
18
19 ALL CODE CONTAINED HEREIN IS PUBLIC
20 DOMAIN AND MAY BE FREELY COPIED FOR
21 NON-COMMERCIAL PURPOSES IF ACKNOWLEDGED
22 APPROPRIATELY.                        )
23 -->
```

```
SCR # 31
 0 ( RADAR/2                SAB 6/83 )
 1
 2 (
 3 THIS PROGRAM IS IN FIG-FORTH AND ASSEM-
 4 BLY LANGUAGE FOR A 6502 MICROPROCESSOR.
 5 IT IS DESIGNED TO RUN ON AN APPLE II OR
 6 II+ MICROCOMPUTER WITH A JOHN BELL
 7 ENGINEERING PARALLEL INTERFACE CARD, A
 8 SCITRONICS REAL-TIME CLOCK CARD, AND A
 9 SINGLE APPLE FLOPPY DISK DRIVE. IT RE-
10 QUIRES TWO 8-BIT A/D CONVERTERS WITH
11 SAMPLE-AND-HOLD CIRCUITS, AND CONVERSION
12 TIMES OF 6 MICROSEC OR BETTER, E.G FROM
13 MICRO NETWORKS INC. START-CONVERT
14 SIGNALS ARE PROVIDED BY THE JBE CARD
15 UNDER PROGRAM CONTROL. A TTL INTERRUPT
16 SIGNAL MUST BE PROVIDED WHICH COINCIDES
17 WITH THE START OF THE TRANSMITTER PULSE.
18 THE PROGRAM PROVIDES COHERENT INTEGRA-
19 TION OF BOTH QUADRATURE CHANNELS AND
20 PROVIDES 1 HOUR OF CORRELATED DATA ON
21 ONE SIDE OF A FLOPPY DISK.            )
22
23 -->
```

```
SCR # 32
 0 ( RADAR/3                ADR 8/84 )
 1
 2 FORTH DEFINITIONS HEX
 3
 4 ( USER CONSTANTS AND VARIABLES )
 5
 6    5 CONSTANT INPUTSLOT
 7    7 CONSTANT CLOCKSLOT
 8 7200 CONSTANT INBUFFER
 9   40 CONSTANT SECTION     ( 8,10,20,40 )
10    2 CONSTANT LAGS     ( 100/SECTION-2 )
11   3C CONSTANT HEIGHTS     ( <= SECTION )
12   14 CONSTANT DISPHTS          ( <= 14 )
13    9 CONSTANT LOWHEIGHT
14   3C CONSTANT MINS/DISK        ( <= 3D )
15  190 VARIABLE SAMPLES          ( <= 1A0 )
16   32 VARIABLE PULSES
17   40 VARIABLE MINS
18   13 VARIABLE DELAY1            ( >= 13 )
19    C VARIABLE DELAY2
20    1 VARIABLE MIN#
21   32 VARIABLE AMPFACTOR
22    A VARIABLE FREQFACTOR
23                              -->
```

```
SCR # 33
 0 ( RADAR/4                SAB 6/83 )
 1
 2 ( SET DERIVED CONSTANTS FOR VIA )
 3
 4 INPUTSLOT C0 + 100 * CONSTANT INADDR
 5
 6 INADDR       CONSTANT DRB1
 7 INADDR 1+    CONSTANT DRA1
 8 INADDR 2+    CONSTANT DDRB1
 9 INADDR 3 +   CONSTANT DDRA1
10 INADDR B +   CONSTANT ACR1
11 INADDR C +   CONSTANT PCR1
12 INADDR E +   CONSTANT IER1
13 INADDR 80 +  CONSTANT DRB2
14 INADDR 82 +  CONSTANT DDRB2
15 INADDR 84 +  CONSTANT T1CL2
16 INADDR 85 +  CONSTANT T1CH2
17 INADDR 86 +  CONSTANT T1LL2
18 INADDR 87 +  CONSTANT T1LH2
19 INADDR 8B +  CONSTANT ACR2
20 INADDR 8C +  CONSTANT PCR2
21 INADDR 8E +  CONSTANT IER2
22
23 -->
```

```
SCR # 34
 0 ( RADAR/5                ADR 8/84 )
 1
 2 ( APPLE II FIXED LOCATIONS )
 3
 4 22 CONSTANT WINDOW
 5 24 CONSTANT CURSOR
 6 45 CONSTANT ACC
 7 3FE CONSTANT IRQVECTORADDR
 8 C030 CONSTANT SPEAKER
 9
10 ( ZERO-PAGE QUEUE-POINTER LOCATIONS )
11
12 52 CONSTANT QUEUE1
13 53 CONSTANT QUEUE2
14 54 CONSTANT HREAL
15 56 CONSTANT HIMAG
16 58 CONSTANT LSUM
17 5A CONSTANT HSUM
18
19 -->
20
21
22
23
```

```
SCR # 35
 0 ( RADAR/6                SAB 6/83 )
 1
 2 ( ZERO-PAGE SCRATCHPAD LOCATIONS )
 3
 4 60 CONSTANT LAG
 5 61 CONSTANT BUFS
 6 62 CONSTANT BUFL
 7 63 CONSTANT BUFH
 8
 9 ( ZERO-PAGE CONSTANTS AND VARIABLES )
10
11 64 CONSTANT INCREM
12 65 CONSTANT INCREM*2
13 66 CONSTANT PULSECOUNT
14 67 CONSTANT SPARES
15 69 CONSTANT SAMPLECOUNT
16
17 -->
18
19
20
21
22
23
```

```
SCR # 36
 0 ( RADAR/7                SAB 6/83 )
 1
 2 ( ZERO-PAGE SAMPLE AND DATA POINTERS ) .
 3
 4 70 CONSTANT LSP1
 5 72 CONSTANT HSP1
 6 74 CONSTANT LSP2
 7 76 CONSTANT HSP2
 8 78 CONSTANT LDP
 9 7A CONSTANT MDP
10 7C CONSTANT HDP
11
12 ( PAGE POINTERS FOR BUFFERS )
13
14 80 CONSTANT INBUF
15 81 CONSTANT LDR
16 82 CONSTANT HDR
17 83 CONSTANT LDI
18 84 CONSTANT HDI
19 85 CONSTANT Z
20
21 -->
22
23
```

```
SCR # 37
 0 ( RADAR/8                SAB 6/83 )
 1
 2 ( PAGE POINTERS CONTINUED )
 3
 4 86 CONSTANT LOR
 5 87 CONSTANT HOR
 6 88 CONSTANT LOI
 7 89 CONSTANT HOI
 8 8A CONSTANT LRR
 9 8B CONSTANT LII
10 8C CONSTANT LRI
11 8D CONSTANT LIR
12 8E CONSTANT MRR
13 8F CONSTANT MII
14 90 CONSTANT MRI
15 91 CONSTANT MIR
16 92 CONSTANT HRR
17 93 CONSTANT HII
18 94 CONSTANT HRI
19 95 CONSTANT HIR
20
21 : ADDR C@ 100 * ;
22
23 -->
```

```
SCR # 38
 0 ( RADAR/9                ADR 8/84 )
 1
 2 ( INITIALIZE ZERO PAGE AND VARIABLES )
 3
 4 : INITLOC                          ( --- )
 5
 6 PULSES @ PULSECOUNT C!
 7 1 MIN# ! 0 SAMPLECOUNT !
 8 SECTION 201 * INCREM !
 9 0 QUEUE2 C! INCREM C@ QUEUE1 C!
10 INBUFFER 100 / 16 0
11 DO DUP I + INBUF I + C!
12 LOOP DROP                ( PAGE POINTERS )
13 INCREM*2 C@
14 DUP LSUM C! HSUM C!
15 HDR C@ HREAL 1+ C!    ( FOR RT DISPLAY )
16 HDI C@ HIMAG 1+ C! ;
17
18 INITLOC
19
20 -->
21
22
23
```

```
SCR # 39
 0 ( RADAR/10              SAB 6/83 )
 1
 2 ( INITIALIZE BUFFERS )
 3
 4 : INITBUF                          ( --- )
 5
 6 INBUFFER DUP 1600 + SWAP
 7 DO 0 I C!
 8 LOOP ;
 9
10 ( INITIALIZE I/O CARD )
11
12 : INITIO                          ( --- )
13
14 0  DDRB1 C! 0  DDRA1 C!
15 3  ACR1 C!  40 PCR1 C!
16 81 DDRB2 C! 80 DRB2 C!
17 0  ACR2 C!  0  PCR2 C!
18 7F IER2 C! ;
19
20 -->
21
22
23
```

```
SCR # 40
 0 ( RADAR/11               ADR 8/84 )
 1
 2 ( CODE TO INPUT TO BUFFER FROM I/O )
 3
 4 : MACRO1
 5
 6 ASSEMBLER HEIGHTS 0
 7 DO INADDR LDA, INBUFFER I + STA,
 8 INADDR 1+ LDA, INBUFFER I 80 + + STA,
 9 NOP, NOP, NOP, LOOP ;
10
11 ( CODE FOR COHERENT INTEGRATION )
12
13 : MACRO2
14
15 ASSEMELER HEIGHTS 0
16 DO LSUM )Y LDA, INBUFFER 1 + ,X ADC,
17   LSUM )Y STA, HSUM )Y LDA, 0 # ADC,
18   HSUM )Y STA, CLC, INY,
19 LOOP ;
20
21 -->
22
23
```

```
SCR # 41
 0 ( RADAR/12               ADR 8/84 )
 1
 2 ( INTERRUPT SERVICE ROUTINE )
 3
 4 CODE INTERRUPT                     ( --- )
 5
 6 TXA, PHA, TYA, PHA,          ( SAVE REGS )
 7 INADDR LDA,           ( RESET INTERRUPT )
 8 C0 # LDA, ACR2 STA,           ( FOR PB7 )
 9 DELAY1 LDA, T1CL2 STA,
10 DELAY1 1+ LDA, T1CH2 STA,  ( SET FUZE )
11 3 # LDA, T1LL2 STA,          ( 100 KHZ )
12 0 # LDY, T1LH2 STY,
13 DELAY2 LDX,               ( * 5 MICROSEC )
14 BEGIN, NOP, NOP, NOP, NOP,
15 NOP, NOP, DEX, 0=             ( NO YET?)
16 END, 80 # LDA, DRB2 STA,    ( PB7 HIGH )
17 INADDR LDA, INADDR 1+ LDA,
18 MACRO1                    ( LOAD BUFFER )
19
20
21 -->
22
23
```

```
SCR # 42
 0 ( RADAR/13              ADR 8/84 )
 1
 2 ( CONTINUATION OF INTERRUPT )
 3
 4 CLC,
 5 LDR LDA, LSUM 1+ STA,
 6 1 # ADC, HSUM 1+ STA, 0 # LDX,
 7 0 # LDY, CLC, MACRO2       ( SUM DATA )
 8 LDI LDA, LSUM 1+ STA,
 9 1 # ADC, HSUM 1+ STA, 80 # LDX,
10 0 # LDY, CLC, MACRO2
11 0 # LDA, ACR2 STA,          ( STOP PB7 )
12 PULSECOUNT DEC, 0=   ( ENOUGH PULSES? )
13 IF, 8 # LDA, IER1 STA,      ( DISABLE )
14 THEN, SPEAKER LDA,          ( AUDIBLE )
15 PLA, TAY, PLA,
16 TAX, ACC LDA,     ( RESTORE REGISTERS )
17 RTI,
18
19 -->
20
21
22
23
```

```
SCR # 43
 0 ( RADAR/14              SAB 6/83 )
 1
 2 ( SYNCHRONIZE TO INTERRUPT )
 3
 4 CODE SYNCHRONIZE                ( --- )
 5
 6 SPARES STY, SPARES 1+ STY,     ( RESET )
 7 BEGIN, SPARES INC, 0=         ( CARRY? )
 8   IF, SPARES 1+ INC,    ( COUNT SPARES )
 9   THEN, PULSECOUNT LDA, 0=
10 END, PULSES LDA,
11 PULSECOUNT STA,               ( RESET )
12 NEXT JMP,
13
14 : MACRO3                        ( --- )
15
16 ASSEMBLER INBUFFER 100 + 4 0
17 DO HEIGHTS 0
18   DO DUP I + 500 + LDA, DUP I + ,X STA,
19   LOOP 100 +
20 LOOP DROP ;
21
22 -->
23
```

```
SCR # 44
 0 ( RADAR/15              SAB 6/83 )
 1
 2 CODE ADVANCE          ( --- FINISHFLAG )
 3
 4 XSAVE STX,             ( SAVE REGISTER )
 5 QUEUE1 LDA, QUEUE2 STA,
 6 CLC, INCREM ADC, QUEUE1 STA,
 7 HREAL STA, HIMAG STA, CLC,
 8 INCREM ADC, LSUM STA,
 9 HSUM STA,              ( ADVANCE QUEUE )
10 TAX, MACRO3 XSAVE LDX,   ( GET OFFSET )
11 SAMPLECOUNT INC, 0=           ( CARRY? )
12 IF, SAMPLECOUNT 1+ INC,  ( COUNT THEM )
13 THEN, SAMPLECOUNT 1+ LDA, ( TEST THEM )
14 SAMPLES 1+ CMP, CS             ( MATCH? )
15 IF, SAMPLECOUNT LDA,
16   SAMPLES CMP, CS        ( END MINUTE? )
17   IF, SAMPLECOUNT
18     DUP STY, 1+ STY,      ( ZERO COUNT )
19     1 # LDA, PUSHOA JMP,   ( SET FLAG )
20   THEN,
21 THEN, TYA, PUSHOA JMP,    ( RESET FLAG )
22
23 -->
```

```
SCR # 45
 0 ( RADAR/16              ADR 8/84 )
 1
 2 ( MAKE A TIME MESSAGE )
 3
 4 CLOCKSLOT 10 * C084 + CONSTANT TLOC
 5 D3C3 VARIABLE TIM
 6
 7 HERE 15 ALLOT 15 A0 FILL AF TIM 5 + C!
 8 AF TIM 8 + C! B6A0 TIM 10 + ! C850 TIM
 9 12 + ! D3D4 TIM 14 + !
10
11 : MACRO4                        ( --- )
12
13 ASSEMBLER TLOC LDA, .A LSR, .A LSR,
14 .A LSR, .A LSR, B0 # ORA, ;
15
16 : MACRO5                      ( V --- )
17
18 ASSEMBLER MACRO4 TIM + STA, DEY,
19 TLOC STY, ;
20
21 -->
22
23
```

```
SCR # 46
 0 ( RADAR/17              SAB 6/83 )
 1
 2 CODE TIMEREAD
 3
 4 F0 # LDA, TLOC 1+ STA,
 5  F # LDA, TLOC STA,
 6 FC # LDA, TLOC 3 + STA,
 7 F4 # LDA, TLOC 1+ STA,
 8  C # LDY, TLOC STY,
 9 9 MACRO5 A MACRO5 3 MACRO5 4 MACRO5
10 6 MACRO5 7 MACRO5
11 TLOC LDA, DEY, TLOC STY, MACRO4
12 F1 # AND, TIM C + STA, DEY, TLOC STY,
13 D MACRO5 E MACRO5 MACRO4 TIM F + STA,
14 F8 # LDA, TLOC 1+ STA,
15  F # LDA, TLOC STA,
16 F8 # LDA, TLOC 3 + STA,
17 FC # LDA, TLOC 1+ STA,
18  F # LDA, TLOC STA, NEXT JMP,
19
20 -->
21
22
23
```

```
SCR # 47
 0 ( RADAR/18              SAB 6/83 )
 1
 2 ( WAIT FOR AN EVEN MINUTE )
 3
 4 : AWAITMIN
 5
 6 TIMEREAD TIM F + C@          ( GET MIN )
 7 BEGIN DUP TIMEREAD
 8   TIM F + C@ -      ( HAS MIN CHANGED? )
 9 END DROP ;
10
11 ( CLEAR OUTPUT ACCUMULATORS )
12
13 : CLEAROUT
14
15 LRR ADDR C00 0 FILL ;
16
17 -->
18
19
20
21
22
23
```

```
SCR # 48
 0 ( RADAR/19              SAB 6/83 )
 1
 2 ( ACCUMULATE VALUES )
 3
 4 CODE VALAC
 5
 6 HEIGHTS 1 - # LDY, ( SET NO. OF HGHTS)
 7 BEGIN, CLC, LSP1 )Y LDA, BUFL STA,
 8   HSP1 )Y LDA, BUFH STA, 0<
 9   IF, FF # LDA, BUFS STA,
10   ELSE, 0 # LDA, BUFS STA,
11   THEN, CLC, LDP )Y LDA, BUFL ADC,
12   LDP )Y STA, MDP )Y LDA, BUFH ADC,
13   MDP )Y STA, HDP )Y LDA, BUFS ADC,
14   HDP )Y STA, DEY, 0<
15 END, NEXT JMP,
16
17 -->
18
19
20
21
22
23
```

```
SCR # 49
 0 ( RADAR/20              SAB 6/83 )
 1
 2 ( ACCUMULATE DIFFERENCES )
 3
 4 CODE DIFAC
 5
 6 XSAVE STX, LAG LDX,
 7 BEGIN, HEIGHTS 1 - # LDY,
 8   BEGIN, SEC, LSP1 )Y LDA, LSP2 )Y SBC,
 9     BUFL STA, HSP1 )Y LDA, HSP2 )Y SBC,
10     BUFH STA, 0< NOT
11     IF, CLC, LDP )Y LDA, BUFL ADC,
12       LDP )Y STA, MDP )Y LDA, BUFH ADC,
13       MDP )Y STA, CS
14       IF, HDP )Y LDA, 0 # ADC,
15         HDP )Y STA,
16       THEN,
17
18 -->
19
20
21
22
23
```

```
SCR # 50
 0 ( RADAR/21              SAB 6/83 )
 1
 2     ELSE, SEC, LDP )Y LDA, BUFL SBC,
 3       LDP )Y STA, MDP )Y LDA, BUFH SBC,
 4       MDP )Y STA, CS
 5       IF, HDP )Y LDA, FF # SBC,
 6         HDP )Y STA,
 7       THEN,
 8     THEN, DEY, 0<
 9   END, CLC, LDP LDA, INCREM ADC,
10   LDP STA, MDP STA, HDP STA, SEC,
11   LSP2 LDA, INCREM SBC, LSP2 STA,
12   HSP2 STA, DEX, 0<
13 END, XSAVE LDX, NEXT JMP,
14
15 -->
16
17
18
19
20
21
22
23
```

```
SCR # 51
 0 ( RADAR/22              SAB 6/83 )
 1
 2 ( SET POINTERS )
 3
 4 CODE RR
 5
 6 QUEUE1 LDA, LSP1 STA, HSP1 STA,
 7 QUEUE2 LDA, LSP2 STA, HSP2 STA,
 8 INCREM*2 LDA, LDP STA, MDP STA,
 9 HDP STA,
10 LDR LDA, LSP1 1+ STA, LSP2 1+ STA,
11 HDR LDA, HSP1 1+ STA, HSP2 1+ STA,
12 LRR LDA, LDP 1+ STA,
13 MRR LDA, MDP 1+ STA,
14 HRR LDA, HDP 1+ STA,
15 LAGS 1 - # LDA, LAG STA,
16 NEXT JMP,
17
18 -->
19
20
21
22
23
```

```
SCR # 52
 0 ( RADAR/23              SAB 6/83 )
 1
 2 CODE II
 3
 4 QUEUE1 LDA, LSP1 STA, HSP1 STA,
 5 QUEUE2 LDA, LSP2 STA, HSP2 STA,
 6 INCREM*2 LDA, LDP STA, MDP STA,
 7 HDP STA,
 8 LDI LDA, LSP1 1+ STA, LSP2 1+ STA,
 9 HDI LDA, HSP1 1+ STA, HSP2 1+ STA,
10 LII LDA, LDP 1+ STA,
11 MII LDA, MDP 1+ STA,
12 HII LDA, HDP 1+ STA,
13 LAGS 1 - # LDA, LAG STA,
14 NEXT JMP,
15
16 -->
17
18
19
20
21
22
23
```

```
SCR # 53
 0 ( RADAR/24              SAB 6/83 )
 1
 2 CODE RI
 3
 4 QUEUE1 LDA, LSP1 STA, HSP1 STA,
 5   LSP2 STA, HSP2 STA,
 6 INCREM LDA, LDP STA, MDP STA, HDP STA,
 7 LDR LDA, LSP1 1+ STA,
 8 HDR LDA, HSP1 1+ STA,
 9 LDI LDA, LSP2 1+ STA,
10 HDI LDA, HSP2 1+ STA,
11 LRI LDA, LDP 1+ STA,
12 MRI LDA, MDP 1+ STA,
13 HRI LDA, HDP 1+ STA,
14 LAGS # LDA, LAG STA,
15 NEXT JMP,
16
17 -->
18
19
20
21
22
23
```

```
SCR # 54

 0 ( RADAR/25              SAB 6/83 )
 1
 2 CODE IR
 3
 4 QUEUE1 LDA, LSP1 STA, HSP1 STA,
 5   LSP2 STA, HSP2 STA,
 6 INCREM LDA, LDP STA, MDP STA, HDP STA,
 7 LDI LDA, LSP1 1+ STA,
 8 HDI LDA, HSP1 1+ STA,
 9 LDR LDA, LSP2 1+ STA,
10 HDR LDA, HSP2 1+ STA,
11 LIR LDA, LDP 1+ STA,
12 MIR LDA, MDP 1+ STA,
13 HIR LDA, HDP 1+ STA,
14 LAGS # LDA, LAG STA,
15 NEXT JMP,
16
17 -->
18
19
20
21
22
23
```

```
SCR # 55

 0 ( RADAR/26              SAB 6/83 )
 1
 2 CODE RZ
 3
 4 QUEUE1 LDA, LSP1 STA, HSP1 STA,
 5 INCREM LDA, LDP STA, MDP STA, HDP STA,
 6 LDR LDA, LSP1 1+ STA,
 7 HDR LDA, HSP1 1+ STA,
 8 Z LDA, LSP2 1+ STA, HSP2 1+ STA,
 9 LRR LDA, LDP 1+ STA,
10 MRR LDA, MDP 1+ STA,
11 HRR LDA, HDP 1+ STA,
12 0 # LDA, LAG STA,
13 NEXT JMP,
14
15 -->
16
17
18
19
20
21
22
23
```

```
SCR # 56

 0 ( RADAR/27              SAB 6/83 )
 1
 2 CODE IZ
 3
 4 QUEUE1 LDA, LSP1 STA, HSP1 STA,
 5 INCREM LDA, LDP STA, MDP STA, HDP STA,
 6 LDI LDA, LSP1 1+ STA,
 7 HDI LDA, HSP1 1+ STA,
 8 Z LDA, LSP2 1+ STA, HSP2 1+ STA,
 9 LII LDA, LDP 1+ STA,
10 MII LDA, MDP 1+ STA,
11 HII LDA, HDP 1+ STA,
12 0 # LDA, LAG STA,
13 NEXT JMP,
14
15 -->
16
17
18
19
20
21
22
23
```

```
SCR # 57

 0 ( RADAR/28              SAB 6/83 )
 1
 2 CODE RV
 3
 4 QUEUE1 LDA, LSP1 STA, HSP1 STA,
 5 0 # LDA, LDP STA, MDP STA, HDP STA,
 6 LDR LDA, LSP1 1+ STA,
 7 HDR LDA, HSP1 1+ STA,
 8 LRR LDA, LDP 1+ STA,
 9 MRR LDA, MDP 1+ STA,
10 HRR LDA, HDP 1+ STA,
11 NEXT JMP,
12
13 -->
14
15
16
17
18
19
20
21
22
23
```

```
SCR # 58

 0 ( RADAR/29              SAB 6/83 )
 1
 2 CODE IV
 3
 4 QUEUE1 LDA, LSP1 STA, HSP1 STA,
 5 0 # LDA, LDP STA, HDP STA, HDP STA,
 6 LDI LDA, LSP1 1+ STA,
 7 HDI LDA, HSP1 1+ STA,
 8 LII LDA, LDP 1+ STA,
 9 MII LDA, MDP 1+ STA,
10 HII LDA, HDP 1+ STA,
11 NEXT JMP,
12
13 -->
14
15
16
17
18
19
20
21
22
23
```

```
SCR # 59

 0 ( RADAR/30              SAB 6/83 )
 1
 2 ( UPDATE OFFSET FOR ZERO MEAN )
 3
 4 MRR ADDR CONSTANT MREGR
 5 LOR ADDR CONSTANT LOFFR
 6 0 VARIABLE OFF
 7 0 VARIABLE REG
 8
 9 ( GENERATE HIGH AND LOW BYTES VH AND
10   VL FROM A VALUE V )
11
12 : SPLIT                  ( V --- VL VH )
13
14 SP@ 1+ C@ >R FF AND R> ;
15
16 -->
17
18
19
20
21
22
23
```

```
SCR # 60

 0 ( RADAR/31              SAB 6/83 )
 1
 2 : FINDOFFSET
 3
 4 2 0
 5 DO HEIGHTS 0
 6   DO J 100 * I + MREGR + REG !
 7     J 200 * I + LOFFR + OFF !
 8     REG @ 400 + C@ 100 *  ( HIGH BYTE )
 9     REG @ C@ +          ( ADD MID BYTE )
10     DUP 7FF <=
11     IF 100 SAMPLES @ */     ( POSITIVE )
12     ELSE MINUS 1+ 100 SAMPLES @ */
13       MINUS
14     THEN OFF @ C@ OFF @
15     100 + C@ 100 * +     ( PREV OFFSET )
16     SWAP - SPLIT
17     OFF @ 100 + C! OFF @ C!
18   LOOP
19 LOOP ;
20
21 -->
22
23
```

```
SCR # 61

 0 ( RADAR/32              ADR 8/84 )
 1
 2 ( MOVE MID AND HI BYTES FOR ODD MIN )
 3
 4 : DISPLACE
 5
 6 MRR ADDR HIR ADDR 100 + 800 CMOVE ;
 7
 8 ( CREATE HEADERFILE )
 9
10 : HEADER
11
12 TIMEREAD TIM 48 BLOCK 30E + 17 CMOVE
13 UPDATE FLUSH ;
14
15 ( GET BLOCK NO. B# FROM FILE NO. F# )
16
17 : BLOCK#                  ( F# --- B# )
18
19 DUP F >
20 IF 3 +
21 THEN 4 * 1+ ;
22
23 -->
```

SCR # 62

```
 0 ( RADAR/33              SAB 6/83 )
 1
 2 ( MOVE 4K BYTES FROM ADDRESS A TO 4
 3   CONTIGUOUS DISK BUFFERS ASSIGNED TO
 4   BLOCKS STARTING WITH BLOCK NO. B# )
 5
 6 : BMOVE                    ( A B# -- )
 7
 8 DUP BUFFER       ( ASSIGN FIRST BUFFER )
 9 5000 OVER !     ( INSERT DOS FILE ADDR )
10 FFA OVER 2+ !    ( AND DOS FILE LENGTH )
11 >R OVER R> 4 +
12 3FC CMOVE UPDATE  ( FILL FIRST BUFFER )
13 4 1                  ( 3 MORE BUFFERS )
14 DO OVER I 400 * + 4 -         ( SOURCE )
15   OVER I + BUFFER       ( DESTINATION )
16   400 CMOVE UPDATE  ( FILL ONE BUFFER )
17 LOOP DROP DROP ;
18
19 -->
20
21
22
23
```

SCR # 63

```
 0 ( RADAR/34              SAB 6/83 )
 1
 2 ( GIVE AUDIBLE WARNING OF DISK CHANGE.
 3   LASTS FOR N SEC )
 4
 5 : ALARM                        ( N -- )
 6
 7 2C * 0
 8 DO 10 0
 9   DO I 0
10     DO
11     LOOP SPEAKER C@ DROP
12   LOOP
13 LOOP ;
14
15 -->
16
17
18
19
20
21
22
23
```

SCR # 64

```
 0 ( RADAR/35              SAB 6/83 )
 1
 2 ( WRITE FILE EACH EVEN MINUTE )
 3
 4 : WRITEFILE
 5
 6 BASE @ A BASE !            ( SAVE BASE )
 7 MIN# @ 2 /MOD SWAP 0=      ( EVEN MIN? )
 8 IF NBR ADDR SWAP BLOCK#  ( FILL BLOCK )
 9   BMOVE ." MINUTE " MIN#
10   @ . ." COMPLETE" CR
11 ELSE DROP MIN# @ 1 =      ( FIRST MIN? )
12   IF HEADER           ( WRITE DATE/TIME )
13   THEN DISPLACE       ( MOVE UP ODD MIN )
14 THEN MINS/DISK 1 -
15 MIN# @ =               ( NEXT-LAST MIN? )
16 IF 5 ALARM                    ( WARNING )
17 THEN MINS/DISK MIN# @ =    ( LAST MIN? )
18 IF 1 MIN# ! CR BELL
19   ." CHANGE DISK NOW!" CR
20 ELSE 1 MIN# +!              ( NEXT MIN )
21 THEN BASE ! ;           ( RESTORE BASE )
22
23 -->
```

SCR # 65

```
 0 ( RADAR/36              SAB 6/83 )
 1
 2 ( REAL-TIME DISPLAY BUFFERS )
 3
 4 0 VARIABLE PREVREAL DISPHTS 2 - ALLOT
 5 0 VARIABLE PREVIMAG DISPHTS 2 - ALLOT
 6
 7 ( INITIALIZE DISPLAY BUFFERS )
 8
 9 : PREVINIT                         ( -- )
10
11 PREVREAL DISPHTS A FILL
12 PREVIMAG DISPHTS A FILL ;
13
14 ( CONVERT SCREEN DISPLAY LINE NO. L#
15   TO SCREEN ADDRESS SA OF START OF
16   LINE )
17
18 : SCREENADDR               ( L# -- SA )
19
20 404 SWAP 8 /MOD 28 * ROT + SWAP
21 80 * + ;
22
23 -->
```

```
SCR # 66
 0 ( RADAR/37              SAB 6/83 )
 1
 2 ( CODE TO PRINT NEW DISPLAY )
 3
 4 : MACRO6                          ( --- )
 5
 6 ASSEMBLER DISPHTS 0
 7 DO I # LDY, HREAL )Y LDA,  ( GET REAL )
 8   SEC, E0 # ORA, 10 # ADC,
 9   IF # AND, CLC, 3 # ADC,    ( LIMITS )
10   PREVREAL I + STA,        ( SAVE REAL )
11   TAX, D2 # LDA,          ( 'R' SYMBOL )
12   DISPHTS I - 2+
13   SCREENADDR ,X STA,     ( POKE SCREEN )
14   HIMAG )Y LDA,            ( GET IMAG )
15   SEC, E0 # ORA, 10 # ADC,
16   IF # AND, CLC, 3 # ADC,    ( LIMITS )
17   PREVIMAG I + STA,       ( SAVE IMAG )
18   TAX, C9 # LDA,          ( 'I' SYMBOL )
19   DISPHTS I - 2+
20   SCREENADDR ,X STA,     ( POKE SCREEN )
21 LOOP ;
22
23 -->
```

```
SCR # 67
 0 ( RADAR/38              SAB 6/83 )
 1
 2 ( CODE TO ERASE PREVICUS DISPLAY )
 3
 4 : MACRO7                          ( --- )
 5
 6 ASSEMBLER DISPHTS 0
 7 DO PREVREAL I + LDX,     ( POSN OF 'R' )
 8   DISPHTS I - 2+
 9   SCREENADDR DUP ,X STA,  ( ERASE 'R' )
10   PREVIMAG I + LDX,      ( POSN OF 'I' )
11   ,X STA,                  ( ERASE 'I' )
12 LOOP ;
13
14 ( DISPLAY DATA )                 ( --- )
15
16 CODE RTDISPLAY
17
18 XSAVE STX, A0 # LDA, MACRO7 MACRO6
19 XSAVE LDX, NEXT JMP,
20
21 -->
22
23
```

```
SCR # 68
 0 ( RADAR/39              ADR 10/84 )
 1
 2 ( CLEAR RT DISPLAY )
 3
 4 CODE CLEARDISPLAY
 5
 6 XSAVE STX, A0 # LDA,
 7 MACRO7 XSAVE LDX,
 8 NEXT JMP,
 9
10
11
12
13
14
15
16
17
18
19
20
21
22 -->
23
```

```
SCR # 69
 0 ( RADAR/40              SAB 6/83 )
 1
 2 ( INTERRUPT ENABLE AND DISABLE )
 3
 4 CODE ENABLE
 5
 6 88 # LDA, IER1 STA, CLI, NEXT JMP,
 7
 8 CODE DISABLE
 9
10 8 # LDA, IER1 STA, NEXT JMP,
11
12 ( CONSTANTS FOR MINUTE DISPLAY )
13
14 MRR ADDR SECTION + CONSTANT AMPL
15 HRR ADDR SECTION + CONSTANT AMPH
16 MRI ADDR SECTION 2 * + CONSTANT RIL
17 HRI ADDR SECTION 2 * + CONSTANT RIH
18 MIR ADDR SECTION 2 * + CONSTANT IRL
19 HIR ADDR SECTION 2 * + CONSTANT IRH
20
21 -->
22
23
```

```
SCR # 70
 0 ( RADAR/41              SAB 6/83 )
 1
 2 ( DISPLAY AMPLITUDE AND FREQUENCY EACH
 3   MINUTE USING CHARACTERS C1 AND C2
 4   RESPECTIVELY )
 5
 6 : DISPLAYAF              ( C1 C2 --- )
 7
 8 CLEARDISPLAY SWAP DISPHTS 0
 9 DO DUP AMPH I + C@ 100 *
10   AMPL I + C@ +              ( GET AMP )
11   AMPFACTOR @ / 3 + 27 MIN   ( ADJUST )
12   DISPHTS I - 2+
13   SCREENADDR + C! OVER  ( POKE SCREEN )
14   RIH I + C@ 100 *
15   RIL I + C@ + IRH I + C@
16   100 * IRL I + C@ + -      ( GET FREQ )
17   FREQFACTOR @ / 14 +
18   27 MIN 3 MAX              ( ADJUST )
19   DISPHTS I - 2+
20   SCREENADDR + C!       ( POKE SCREEN )
21 LOOP DROP DROP ;
22
23 -->
```

```
SCR # 71
 0 ( RADAR/42              SAB 6/83 )
 1
 2 ( WRITE KM LABELS )
 3
 4 : KMLABEL                        ( --- )
 5
 6 CLEAR BASE @ A BASE ! CR CR CR
 7 LOWHEIGHT DUP DISPHTS 3 * 2 / + 2 -
 8 DO I . CR CR -3
 9 +LOOP BASE ! ;
10
11 ( INITIALIZE AT START OF RUN )
12
13 : INIT                           ( --- )
14
15 ' INTERRUPT IRQVECTORADDR !
16 INITBUF INITLOC INITIO
17 PREVINIT CLEAR KMLABEL
18 0 CURSOR ! CR 300 WINDOW ! ;
19
20 -->
21
22
23
```

```
SCR # 72
 0 ( RADAR/43              SAB 6/83 )
 1
 2 ( FILL QUEUE WITH DATA )
 3
 4 : FILLQUEUE                      ( --- )
 5
 6 SAMPLES @ LAGS - 2 - SAMPLECOUNT !
 7 ENABLE
 8 BEGIN SYNCHRONIZE ADVANCE ENABLE
 9 END DISABLE ;
10
11 ( CORRELATE 1 MIN OF DATA )
12
13 : CORRELATE                      ( --- )
14
15 ENABLE
16 BEGIN RV VALAC IV VALAC RZ DIFAC
17   IZ DIFAC RR DIFAC II DIFAC
18   RI DIFAC IR DIFAC RTDISPLAY
19   SYNCHRONIZE ADVANCE ENABLE
20 END DISABLE ;
21
22 -->
23
```

```
SCR # 73
 0 ( RADAR/44              ADR 10/84 )
 1
 2 ( MULTIPLY EVERY OUTBUFFER BY 10H )
 3
 4 : FIXSTRA1                       ( --- )
 5
 6 4 0
 7 DO                    ( FOR RR II RI IR )
 8   LAGS 2 + 1
 9   DO                    ( FOR EACH LAG )
10     HRR J + ADDR
11     SECTION I * +
12     DUP 14 + SWAP
13     DO                ( FOR LOWER  HTS. )
14       I C@ 100 *       ( MULTIPLY EACH )
15       I 400 - C@ + 10 *   ( 3 BYTE NO. )
16       I 800 - C@ 10 / +       ( BY 10H )
17       100 /MOD
18       I C! I 400 - C!
19     LOOP
20   LOOP
21 LOOP ;
22
23 -->
```

SCR # 74

```
 0 ( RADAR/43              ADR 8/84 )
 1
 2 ( RUN PROGRAM )
 3
 4 : GO                                ( ---- )
 5
 6 DISABLE INIT AWAITMIN            ( SET UP )
 7 FILLQUEUE CORRELATE          ( DUMMY MIN )
 8 FINDOFFSET               ( INITIAL OFFSET )
 9 BEGIN AWAITMIN           ( AWAIT EVEN MIN )
10   A0 A0 DISPLAYAF             ( ERASE A,F )
11   CLEAROUT                ( CLEAR OUTPUTS )
12   FILLQUEUE                  ( LOAD QUEUE )
13   CORRELATE             ( CORRELATE 1 MIN )
14   FINDOFFSET            ( REFIGURE OFFSET )
15   C1 C6 DISPLAYAF             ( PRINT A,F )
16   EMPTY-BUFFERS       ( CLEAR DISK BUFFS )
17   WRITEFILE FLUSH            ( DISK WRITE )
18   MIN# @ MINS @ =            ( RUN ENDED? )
19 END ;
20
21 DECIMAL ;S
22
23
```

SCR # 75

```
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

APPENDIX B
Screens from Previous Collection Program

```
SCR # 30

 0 ( RADAR/1                SAB 6/83 )
 1
 2 (
 3 COHERENT-SCATTER-RADAR DATA-ACQUISITION
 4 PROGRAM WRITTEN OCTOBER 1981 BY SIDNEY
 5 BOWHILL, REVISED JUNE 1983. RESEARCH
 6 SUPPORTED BY NATIONAL SCIENCE FOUNDATION
 7 AND BY NATIONAL AERONAUTICS AND SPACE
 8 ADMINISTRATION.
 9
10 ASSISTANCE OF FRANCIS KEASLER AND DAVID
11 PADGITT IS ACKNOWLEDGED. ORIGINAL
12 FIG-FORTH SYSTEM BY FORTH INTEREST
13 GROUP, SAN CARLOS, CALIFORNIA.
14
15 ASSEMBLER, EDITOR AND METACOMPILER BY
16 GEORGE LYONS, JERSEY CITY NJ.
17
18 ALL CODE CONTAINED HEREIN IS PUBLIC
19 DOMAIN AND MAY BE FREELY COPIED FOR
20 NON-COMMERCIAL PURPOSES IF ACKNOWLEDGED
21 APPROPRIATELY.                        )
22
23 -->
```

```
SCR # 32

 0 ( RADAR/3                SAB 6/83 )
 1
 2 FORTH DEFINITIONS HEX
 3
 4 ( USER CONSTANTS AND VARIABLES )
 5
 6    2 CONSTANT INPUTSLOT
 7    3 CONSTANT CLOCKSLOT
 8 6200 CONSTANT INBUFFER
 9   20 CONSTANT SECTION     ( 8,10,20,40 )
10    6 CONSTANT LAGS      ( 100/SECTION-2 )
11   14 CONSTANT HEIGHTS     ( <= SECTION )
12   14 CONSTANT DISPHTS          ( <= 14 )
13   3C CONSTANT LOWHEIGHT
14   3C CONSTANT MINS/DISK        ( <= 3D )
15  190 VARIABLE SAMPLES         ( <= 1A0 )
16   32 VARIABLE PULSES
17   40 VARIABLE MINS
18   13 VARIABLE DELAY1           ( >= 13 )
19   42 VARIABLE DELAY2            ( >= 1 )
20    1 VARIABLE MIN#
21   32 VARIABLE AMPFACTOR
22   64 VARIABLE FREQFACTOR
23                              -->
```

```
SCR # 34

 0 ( RADAR/5                SAB 6/83 )
 1
 2 ( APPLE II FIXED LOCATIONS )
 3
 4 22 CONSTANT WINDOW
 5 24 CONSTANT CURSOR
 6 38 CONSTANT KSWL
 7 39 CONSTANT KSWH
 8 45 CONSTANT ACC
 9 3FE CONSTANT IRQVECTORADDR
10 C030 CONSTANT SPEAKER
11
12 ( ZERO-PAGE QUEUE-POINTER LOCATIONS )
13
14 50 CONSTANT PORT
15 52 CONSTANT QUEUE1
16 53 CONSTANT QUEUE2
17 54 CONSTANT HREAL
18 56 CONSTANT HIMAG
19 58 CONSTANT LSUM
20 5A CONSTANT HSUM
21
22 -->
23
```

```
SCR # 38

 0 ( RADAR/9                SAB 6/83 )
 1
 2 ( INITIALIZE ZERO PAGE AND VARIABLES )
 3
 4 : INITLOC                          ( --- )
 5
 6 DRB1 PORT !
 7 PULSES @ PULSECOUNT C!
 8 1 MIN# ! 0 SAMPLECOUNT !
 9 SECTION 201 * INCREM !
10 0 QUEUE2 C! INCREM C@ QUEUE1 C!
11 INBUFFER 100 / 16 0
12 DO DUP I + INBUF I + C!
13 LOOP DROP              ( PAGE POINTERS )
14 INCREM*2 C@
15 DUP LSUM C! HSUM C!
16 HDR C@ HREAL 1+ C!    ( FOR RT DISPLAY )
17 HDI C@ HIMAG 1+ C! ;
18
19 INITLOC
20
21 -->
22
23
```

```
SCR # 40

 0 ( RADAR/11              SAB 6/83 )
 1
 2 ( CODE TO INPUT TO BUFFER FROM I/O )
 3
 4 : MACRO1
 5
 6 ASSEMBLER HEIGHTS 0
 7 DO PORT )Y LDA, INBUFFER I + ,X STA,
 8 LOOP ;
 9
10 ( CODE FOR COHERENT INTEGRATION )
11
12 : MACRO2
13
14 ASSEMBLER HEIGHTS 0
15 DO LSUM )Y LDA, INBUFFER I + ,X ADC,
16   LSUM )Y STA, HSUM )Y LDA, 0 # ADC,
17   HSUM )Y STA, INY,
18 LOOP ;
19
20 -->
21
22
23
```

```
SCR # 41

 0 ( RADAR/12              SAB 6/83 )
 1
 2 ( INTERRUPT SERVICE ROUTINE )
 3
 4 CODE INTERRUPT                    ( --- )
 5
 6 TXA, PHA, TYA, PHA,          ( SAVE REGS )
 7 INADDR LDA,          ( RESET INTERRUPT )
 8 C0 # LDA, ACR2 STA,           ( FOR PB7 )
 9 DELAY1 LDA, T1CL2 STA,
10 DELAY1 1+ LDA, T1CH2 STA,   ( SET FUZE )
11 3 # LDA, T1LL2 STA,           ( 100 KHZ )
12 0 # LDY, T1LH2 STY,
13 DELAY2 LDX,               ( * 5 MICROSEC )
14 BEGIN, DEX, 0=                ( NO YET? )
15 END, 80 # LDA, DRB2 STA,    ( PB7 HIGH )
16 CLC, PORT LDA,
17 .A ROR, .A ROR, TAX,     ( X FROM PORT )
18 PORT )Y LDA,          ( RESET ADC LATCH )
19 INBUFFER ,X STA,    ( TIME FOR ADC EOC )
20 MACRO1                   ( LOAD BUFFER )
21 PORT LDA, DRB2 STA,        ( FOR SCOPE )
22
23 -->
```

```
SCR # 42

 0 ( RADAR/13              SAB 6/83 )
 1
 2 ( CONTINUATION OF INTERRUPT )
 3
 4 LDR LDA, PORT ADC,
 5 PORT ADC, LSUM 1+ STA,  ( SET LSUM HI )
 6 1 # ADC, HSUM 1+ STA,   ( SET HSUM HI )
 7 0 # LDY, CLC, MACRO2        ( SUM DATA )
 8 PORT LDA, 1 # EOR,
 9 PORT STA,              ( SWITCH INPUTS )
10 0 # LDA, ACR2 STA,          ( STOP PB7 )
11 PULSECOUNT DEC, 0=   ( ENOUGH PULSES? )
12 IF, 8 # LDA, IER1 STA,       ( DISABLE )
13 THEN, SPEAKER LDA,           ( AUDIBLE )
14 PLA, TAY, PLA,
15 TAX, ACC LDA,      ( RESTORE REGISTERS )
16 RTI,
17
18 -->
19
20
21
22
23
```

```
SCR # 45

 0 ( RADAR/16              SAB 6/83 )
 1
 2 ( MAKE A TIME MESSAGE )
 3
 4 CLOCKSLOT 10 * C084 + CONSTANT TLOC
 5 D3C3 VARIABLE TIM
 6 HERE F ALLOT F A0 FILL AF TIM 5 + C!
 7 AF TIM 8 + C!
 8
 9 : MACRO4                          ( --- )
10
11 ASSEMBLER TLOC LDA, .A LSR, .A LSR,
12 .A LSR, .A LSR, B0 # ORA, ;
13
14 : MACRO5                        ( V --- )
15
16 ASSEMBLER MACRO4 TIM + STA, DEY,
17 TLOC STY, ;
18
19 -->
20
21
22
23
```

```
SCR # 61
 0 ( RADAR/32              SAB 6/83 )
 1
 2 ( MOVE MID AND HI BYTES FOR ODD MIN )
 3
 4 : DISPLACE
 5
 6 MRR ADDR HIR ADDR 100 + 800 CMOVE ;
 7
 8 ( CREATE HEADERFILE )
 9
10 : HEADER
11
12 TIMEREAD TIM 48 BLOCK 30E + 11 CMOVE
13 UPDATE FLUSH ;
14
15 ( GET BLOCK NO. B# FROM FILE NO. F# )
16
17 : BLOCK#                  ( F# --- B# )
18
19 DUP F >
20 IF 3 +
21 THEN 4 * 1+ ;
22
23 -->
```

```
SCR # 72
 0 ( RADAR/43              SAB 6/83 )
 1
 2 ( RUN PROGRAM )
 3
 4 : GO                              ( --- )
 5
 6 DISABLE INIT AWAITMIN          ( SET UP )
 7 FILLQUEUE CORRELATE         ( DUMMY MIN )
 8 FINDOFFSET              ( INITIAL OFFSET )
 9 BEGIN AWAITMIN          ( AWAIT EVEN MIN )
10   A0 A0 DISPLAYAF           ( ERASE A,F )
11   CLEAROUT               ( CLEAR OUTPUTS )
12   FILLQUEUE                 ( LOAD QUEUE )
13   CORRELATE            ( CORRELATE 1 MIN )
14   FINDOFFSET           ( REFIGURE OFFSET )
15   C1 C6 DISPLAYAF           ( PRINT A,F )
16   EMPTY-BUFFERS      ( CLEAR DISK BUFFS )
17   WRITEFILE FLUSH          ( DISK WRITE )
18   MIN# @ MINS @ =          ( RUN ENDED? )
19 END ;
20
21 DECIMAL ;S
22
23
```

APPENDIX C
Listing of ANAL4

```
1  REM  !INTEGERBA,H,FI,X1,X2,X3,AX,BX,AD,H1,I,J,K,V(4),A(4,4,20),R(20,2)

2  REM :ANAL4 2/14/83
10 PI = 3.14159: HIMEM: 14000
20 K1 =  - 600 / (PI * 40.92): REM UP IS DOWN!
50  DIM A(4,4,20),R(20,2),V(4)
100 BA = 15000
102  INPUT "NO. OF FILES: ";FI
104  FOR H = 1 TO FI: PRINT "BLOADFILE "H",A23488"
106  FOR H1 = 25536 TO 23488 STEP  - 2048
110  FOR I = 0 TO 3: FOR J = 1 TO 3
120 AX = H1 + 256 * I + 32 * J
130 BX = H1 + 1024 + 256 * I + 32 * J
133  FOR K = 0 TO 19
140 A(I,J,K) =  PEEK (AX) + 256 *  PEEK (BX)
145 AX = AX + 1:BX = BX + 1
150  NEXT : NEXT : NEXT
155 M = (H - 1) * 2 + (25536 - H1) / 2048
160  FOR J = 0 TO 19
170 C1 = ((A(0,1,J)) ^ 2) * 2
180 C2 = ((A(1,1,J)) ^ 2) * 2
187  FOR I = 2 TO 3
190 D1 = (A(0,I,J)) ^ 2
200 D2 = (A(1,I,J)) ^ 2
210 D3 = (A(2,I,J)) ^ 2
220 D4 = (A(3,I,J)) ^ 2
230 E = (C1 - D1 + C2 - D2) / 2
240 F = (D3 - D4) / 2
250 G = (E * E + F * F) ^ .5
251 G(I) = G - 264
252  IF G = 0 THEN G = .001
254 S = F / G:C = E / G: GOSUB 1000
256 V(I) =  INT (V * 10 / (I - 1) + .5)
260  NEXT
265 X1 =  INT (21.72 *  LOG ((C1 + C2) / 10000) + .5)
267  IF X1 < 0 THEN X1 = X1 + 256
270 AD = BA + 60 * M + 3 * J
272  IF G(2) < (C1 + C2) / 20 THEN X2 = 128:X3 = 128: GOTO 280
273 R(J,1) = R(J,1) + 1: IF (G(3) < 1 OR G(2) =  < G(3)) THEN X2 = 128:X3
     = V(2): GOTO 278
274 R(J,2) = R(J,2) + 1:X2 =  INT (6.25 * (3 *  LOG (2) /  LOG (G(2) / G(
    3))) ^ .5 + .5): IF X2 = 128 THEN X2 = 129
275  IF X2 > 200 THEN X2 = 200
276 X3 = V(2): IF X3 > 127 THEN X3 = 128
277  IF X2 > 12 THEN X3 = V(3)
278  IF X3 < 0 THEN X3 = X3 + 256
280  PRINT X1,X2,X3
285  POKE AD,X1: POKE AD + 1,X2: POKE AD + 2,X3
287  NEXT : PRINT "MINUTE "M" COMPLETED"
288  NEXT : NEXT
290  FOR J = BA + FI * 120 TO BA + 3600
295  POKE J,128: NEXT
300  INPUT "FILE NAME?";A$
```

```
310  PRINT "BSAVE"A$",A15000,L3600"
320  FOR I = 0 TO 19: PRINT I;" ";FI * 2;" ";R(I,1);" ";R(I,2): NEXT
330  END
1000  IF C = 0 THEN C = .0001
1010 T = S / C:V =  ATN (T)
1050  IF C < 0 THEN V = V - PI
1060  IF V <  - PI THEN V = V + PI + PI
1210 V = V * K1
1220  RETURN
```

APPENDIX D
Listing of CONVERT84.1

```
1  REM  !INTEGERP1,P2,C1,C2,V1,V2,              P,C,V,I,J,BA,HM,AD,XL,F,X,F
     ,P4,C4,V4
10  REM CONVERT BINARY TO ASCII
20  REM    S BOWHILL 2/14/83
100  REM MAIN PROGRAM
110 BA = 20000
120  HIMEM: 19000
130  INPUT "INPUT FILE NAME: ";F$
135 HM = 60
140  PRINT "BLOAD"F$",A20000"
150  GOSUB 1000
160  GOSUB 2000
170 T$ =  STR$ ( INT (H * 100 + M + .5))
175 D$ =  CHR$ (4)
180  PRINT D$"OPENPOWW/"T$
190  PRINT D$"WRITEPOWW/"T$
193 N$ = " POWER (LOG PLOT)"
195  GOSUB 4000
197  PRINT .02 * P2: PRINT .02 * P1: PRINT .02 * P3: PRINT HM
200 AD = BA:XL = 200:F = 2
210  GOSUB 3000
220  PRINT D$"CLOSEPOWW/"T$
230  PRINT D$"OPENCORR/"T$
240  PRINT D$"WRITECORR/"T$
243 N$ = " CORRELATION TIME (SEC)"
245  GOSUB 4000
247  PRINT .1 * C2: PRINT .1 * C1: PRINT .1 * C3: PRINT HM
250 AD = BA + 1:XL = 255:F = 2
260  GOSUB 3000
270  PRINT D$"CLOSECORR/"T$
280  PRINT D$"OPENVELL/"T$
290  PRINT D$"WRITEVELL/"T$
293 N$ = " VELOCITIES (M/S)"
295  GOSUB 4000
297  PRINT .1 * V2: PRINT .1 * V1: PRINT .1 * V3: PRINT HM
300 AD = BA + 2:XL = 128:F = 10
310  GOSUB 3000
320  PRINT D$"CLOSEVELL/"T$
330  END
1000  REM GET HEADER
1001  REM
1010  INPUT "MONTH: ";M$
1020  INPUT "DAY: ";D$
1025  INPUT "YEAR: ";Y$
1030 H$ = M$ + " " + D$ + ", " + Y$
1040  INPUT "HOUR: ";H
1050  INPUT "MINUTE: ";M
1060  INPUT "NUMBER OF RECORDS: ";N3
1070  RETURN
2000  REM MAX, MIN AND MEAN
2001  REM
2010 P1 =  -55:P2 = 200:P3 = 0:P4 = 0
2020 C1 = 0:C2 = 127:C3 = 0:C4 = 0
```

```
2030 V1 =  - 127:V2 = 127:V3 = 0:V4 = 0
2040  FOR I = BA TO BA + 3597 STEP 3
2050 P =  PEEK (I):C =  PEEK (I + 1):V =  PEEK (I + 2)
2060  IF P = 128 THEN 2180
2065  IF P > 200 THEN P = P - 256
2070  IF P > P1 THEN P1 = P
2080  IF P < P2 THEN P2 = P
2090 P3 = P3 + P:P4 = P4 + 1
2100  IF C = 128 THEN 2180
2110  IF C > C1 THEN C1 = C
2120  IF C < C2 THEN C2 = C
2130 C3 = C3 + C:C4 = C4 + 1
2140  IF V > 128 THEN V = V - 256
2150  IF V > V1 THEN V1 = V
2160  IF V < V2 THEN V2 = V
2170 V3 = V3 + V:V4 = V4 + 1
2180  NEXT I
2190 P3 = P3 / P4:C3 = C3 / C4:V4 = V3 / V4
2200  RETURN
3000  REM WRITE DATA
3001  REM
3010  FOR I = 0 TO 19
3020  FOR J = 0 TO 119
3030 X =  PEEK (AD + 3 * I + 60 * J)
3040  IF X = 128 THEN X = 0
3050  IF X > XL THEN X = X - 256
3060  PRINT F * X
3070  NEXT : NEXT
3080  RETURN
4000  REM WRITE HEADER
4001  REM
4005 I$ = "APPLE " + H$ + N$
4010  PRINT I$: PRINT H: PRINT M: PRINT N3
4020  RETURN
```

APPENDIX E
Listing of Analysis Program

```
10  REM !INTEGERINFILEADDR,OUTFILEADDR,SECTIONSIZE,NUMFILES,POW,THRESHOLD
     ,CRRTIME,VEL,INARRAY(4,4,20),ENOUGHPOWER(20),GOODSHAPE(20),VEL(4),MA
     GN(4),REFILE,FILE,HOUR,MINUTE,NMREC
15  REM   !INTEGER  P1MAX,P2MN,P3MEAN,P4COUNT,C1MAX,C2MN,C3MEAN,C4COUNT,V
     1MAX,V2MN,V3MEAN,V4COUNT,I,J,ADDR,VA,TEMPADDR,BASEHEIGHT
20  REM
21  REM ********************
22  REM *     PROCESS      *
23  REM *                  *
24  REM * COHERENT SCATTER *
25  REM *  DATA ANALYSIS   *
26  REM *     PROGRAM      *
27  REM *                  *
28  REM *  TONY RENNIER    *
29  REM *                  *
30  REM * AUGUST 22, 1984  *
31  REM ********************
32  REM
34  REM ***************************
35  REM *  THIS PROGRAM READS     *
36  REM *  BOTH SIDES OF THE      *
37  REM *    DISK AT ONCE         *
38  REM ***************************
39  REM
40  REM ***************************
41  REM *  SELECT HEIGHT RANGE    *
42  REM ***************************
49 HIMEM = 19799
50  HOME
55  PRINT "SELECT (1) 9-37.5 KM"
56  PRINT "        (2) 39-67.5 KM"
57  INPUT "        (3) 69-97.5 KM";TEMP
60 DISP = 20 * (TEMP - 1)
65  IF TEMP = 1 THEN BASEHEIGHT = 7.5:SCLE = 1: GOTO 85
70  IF TEMP = 2 THEN BASEHEIGHT = 37.5:SCLE = 1: GOTO 85
75  IF TEMP = 3 THEN BASEHEIGHT = 67.5:SCLE = 1: GOTO 85
80  GOTO 50
85 BASEHEIGHT$ = "/" +  STR$ (BASEHEIGHT)
100  REM
101  REM ********************
102  REM * DEFINE CONSTANTS *
103  REM *    AND ARRAYS    *
104  REM ********************
105  REM
110 PI = 3.14159
120 MHZ = 40.92
130 K1 =  - 60000 / (PI * MHZ) / SCLE
131 K3 = 43.44 / SCLE
132  REM
140  REM MINUS IS BECAUSE WE
141  REM SHOW + VELOCITIES TO
142  REM BE DOWNWARD FOR
143  REM HISTORICAL REASONS
```

```
144  REM
150 K2 = 12.5 / SCLE * (3 *  LOG (2)) ^ .5
160 D$ =  CHR$ (4): REM CNTRL-D
170 THRESHHOLD = 20
200  DIM INARRAY(4,4,20)
210  DIM ENOUGHPOWER(20)
220  DIM GOODSHAPE(20)
230  DIM VEL(4)
240  DIM MAGN(4)
1000  REM
1001  REM **************
1002  REM * FILL INPUT *
1003  REM *    ARRAY   *
1004  REM **************
1005  REM
1010  REM INPUT FILE CONSISTS OF
1011  REM 2 MINUTES OF DATA,
1012  REM EACH 2048 BYTES LONG,
1013  REM WITH THE EARLIER DATA
1014  REM SECOND IN THE FILE.
1015  REM EACH MINUTE FILE IS 8
1016  REM PAGES OF DATA, IN THE
1017  REM FOLLOWING ORDER:
1018  REM MRR, MII, MRI, MIR,
1019  REM HRR, HII, HRI, HIR.
1020  REM THE VALUE OF RR IS
1021  REM (MRR+256*HRR), ETC.
1022  REM
1100 INFILEADDR = 34300
1110 OUTFILEADDR = 19800
1120 ADDR = OUTFILEADDR
1130 SECTSIZE = 64
1134  INPUT "NO. OF FILES: ";NUMFILES
1135  INPUT "SIDE A ?";DUMMY$
1200  REM
1201  REM FOR EACH OF (USUALLY)
1202  REM 60 FILES
1203  REM
1210  FOR REFILE = 1 TO NUMFILES
1212  IF REFILE = 31 THEN  PRINT "": INPUT "SIDE B ?";DUMMY$
1214 FILE = REFILE
1216  IF REFILE > 30 THEN FILE = FILE - 30
1220 : PRINT D$"BLOADFILE "FILE",A"INFILEADDR
1300  REM
1301  REM FIRST MIN THEN SECOND
1302  REM
1310 : FOR MIN = 0 TO 1
1400  REM
1401  REM RR, II, RI AND IR
1402  REM
1410 :: FOR QUAN = 0 TO 3
1500  REM
1501  REM USE 3 LAGS, AND
```

```
1502  REM CALCULATE ADDRESSES
1503  REM OF HIGH AND LOW BYTES
1504  REM
1510 ::: FOR LAG = 1 TO 3
1520 ::::LOWBYTEADDR = 256 * QUAN + SECTSIZE * LAG + INFILEADDR + 2048 *
     (1 - MIN) + DISP
1530 ::::HIGHBYTEADDR = LOWBYTEADDR + 1024
1600  REM
1601  REM FOR 20 HEIGHTS
1602  REM
1610 :::: FOR HEIGHT = 0 TO 19
1620 :::::INARRAY(QUAN,LAG,HEIGHT) =  PEEK (LOWBYTEADDR) + 256 *  PEEK (H
     IGHBYTEADDR)
1630 :::::LOWBYTEADDR = LOWBYTEADDR + 1
1640 :::::HIGHBYTEADDR = HIGHBYTEADDR + 1
1650 :::: NEXT HEIGHT
1660 ::: NEXT LAG
1670 :: NEXT QUAN
2000  REM
2001  REM **********************
2002  REM * FIND CORRELATIONS *
2003  REM *  AND PHASES FOR   *
2004  REM * 1 MINUTE OF DATA  *
2005  REM **********************
2006  REM
2007  HOME
2010 :: FOR HEIGHT = 0 TO 19
2020 :::RMSQUARED = ((INARRAY(0,1,HEIGHT)) ^ 2) * 2
2030 :::IMSQUARED = ((INARRAY(1,1,HEIGHT)) ^ 2) * 2
2040 ::: IF RMSQUARED = 0 THEN RMSQUARED = .001
2100  REM
2101  REM FIND SIN AND COS
2102  REM COMPONENTS OF
2103  REM COVARIANCE
2104  REM
2110 ::: FOR LAG = 2 TO 3
2120 ::::RRSQUARED = (INARRAY(0,LAG,HEIGHT)) ^ 2
2130 ::::IISQUARED = (INARRAY(1,LAG,HEIGHT)) ^ 2
2140 ::::RISQUARED = (INARRAY(2,LAG,HEIGHT)) ^ 2
2150 ::::IRSQUARED = (INARRAY(3,LAG,HEIGHT)) ^ 2
2160 ::::COCOMPT = (RMSQUARED - RRSQUARED + IMSQUARED - IISQUARED) / 2
2170 ::::SICOMPT = (RISQUARED - IRSQUARED) / 2
2180 ::::MAGN = (COCOMPT ^ 2 + SICOMPT ^ 2) ^ .5
2185  IF MAGN = 0 THEN MAGN = .001
2190 ::::MAGN(LAG) = MAGN
2200  REM
2201  REM FIND PHASE ANGLE
2202  REM AND VELOCITY
2203  REM
2210 :::: IF COCOMPT = 0 THEN COCOMPT = .001
2220 ::::ANGLE =  ATN (SICOMPT / COCOMPT)
2230 :::: IF COCOMPT < 0 THEN ANGLE = ANGLE - PI
2240 :::: IF ANGLE <  - PI THEN ANGLE = ANGLE + 2 * PI
2250 ::::VEL(LAG) =  INT (K1 * ANGLE / (LAG - 1) + .5)
2260 ::: NEXT LAG
2300  REM
```

```
2301  REM FIND POWER AND
2302  REM  MAKE IT A TWO-BYTE
2303  REM SIGNED INTEGER
2304  REM
2310 :::POW =  INT (K3 *  LOG ((RMSQUARED + IMSQUARED) / 1000) + .5)
2320 ::: IF POW > 32767 THEN POW = 32767
2330 ::: IF POW <  - 32767 THEN POW =  - 32767
2340 ::: IF POW < 0 THEN POW = POW + 65536
2400  REM
2401  REM TEST TO SEE IF
2402  REM ENOUGH POWER TO
2403  REM CALCULATE
2404  REM CORRELATION TIME
2405  REM AND VELOCITY
2406  REM
2410 ::: IF ((RMSQUARED + IMSQUARED) / MAGN(2)) > THRESHHOLD THEN CRRTIME
     = 0:VEL = 0: GOTO 3000
2420 :::ENOUGHPOWER(HEIGHT) = ENOUGHPOWER(HEIGHT) + 1
2500  REM
2501  REM CHECK SHAPE OF
2502  REM COVARIANCE: USE
2503  REM FIRST LAG IF
2504  REM  SECOND LAG < .2*FIRST LAG
2505  REM OR > FIRST LAG
2506  REM
2510 ::: IF (MAGN(3) < MAGN(2) / 5 OR MAGN(3) >  = MAGN(2)) THEN CRRTIME =
    0:VEL = VEL(2): GOTO 2740
2520 :::GOODSHAPE(HEIGHT) = GOODSHAPE(HEIGHT) + 1
2600  REM
2601  REM CALCULATE CORRELATION
2602  REM TIME AS UNSIGNED
2603  REM INTEGER, AVOIDING
2604  REM  THE VALUE 32768
2605  REM AND ASSUMING FIRST
2606  REM TWO LAGS LIE ON A
2607  REM GAUSSIAN CURVE
2608  REM
2610 :::CRRTIME =  INT (K2 / ( LOG (MAGN(2) / MAGN(3))) * .5 + .5)
2625 ::: IF CRTIME > 32767 THEN CRTIME = 32767
2700  REM
2701  REM USE FIRST LAG FOR
2702  REM VELOCITY UNLESS
2703  REM CORRELATION TIME IS
2704  REM > 1.2 SEC; THEN
2705  REM SECOND LAG IS USED.
2706  REM VELOCITY IS THEN
2707  REM  MADE A DOUBLE-BYTE
2708  REM SIGNED INTEGER IN
2709  REM  THE RANGE -32767 TO 32767
2710  REM
2720 :::VEL = VEL(2)
2730 ::: IF CRRTIME > 24 THEN VEL = VEL(3)
2740 ::: IF VEL > 32767 THEN VEL = 32767
2750 ::: IF VEL <  - 32767 THEN VEL =  - 32767
2760 ::: IF VEL < 0 THEN VEL = VEL + 65536
2770 ::: GOTO 3000
```

```
3000  REM
3001  REM ********************
3002  REM * POKE POW, CRRTIME*
3003  REM *   AND VEL INTO   *
3004  REM *   OUTPUT FILE    *
3005  REM ********************
3006  REM
3010 ::: PRINT POW,CRRTIME,VEL
3012 :::T1 =  INT (POW / 256)
3014 :::T2 =  INT (CRRTIME / 256)
3016 :::T3 =  INT (VEL / 256)
3020 ::: POKE ADDR,(POW - T1 * 256):ADDR = ADDR + 1
3030 ::: POKE ADDR,T1:ADDR = ADDR + 1
3040 ::: POKE ADDR,(CRRTIME - T2 * 256):ADDR = ADDR + 1
3050 ::: POKE ADDR,T2:ADDR = ADDR + 1
3060 ::: POKE ADDR,(VEL - T3 * 256):ADDR = ADDR + 1
3070 ::: POKE ADDR,T3:ADDR = ADDR + 1
3080 :: NEXT HEIGHT
3090 : NEXT MIN
3100 : PRINT "FILE NO. "REFILE" COMPLETED"
3130  NEXT REFILE
3200  REM
3201  REM FILL MISSING DATA
3202  REM  AREAS WITH 0  S
3203  REM
3210  IF NUMFILES = 60 THEN 4000
3220  FOR ADDR = OUTFILEADDR + NUMFILES * 120 * 2 TO OUTFILEADDR + 14400
3230 : POKE ADDR,0
3240  NEXT ADDR
4000  REM
4001  REM ********************
4002  REM * SAVE OUTPUT FILE *
4003  REM *     TO DISK      *
4004  REM ********************
4005  REM
4040  INPUT "TEXT DISK ?";DUMMY$
4043  REM
4044  REM ***********************
4045  REM * GET HEADER AND FIND*
4046  REM * MIN, MAX, AND MEAN *
4047  REM ***********************
4048  REM
4050  GOSUB 4330
4055  INPUT "MAKE CHANGES?";DUMMY$
4056  IF  LEN (DUMMY$) = 0 THEN 4060
4057  IF  LEFT$ (DUMMY$,1) = "Y" THEN 4050
4060  GOSUB 4430
4062  REM
4070 TME$ =  STR$ ( INT (HOUR * 100 + MINUTE + .5))
4073  REM
4074  REM ********************
4075  REM * WRITE POWER FILE *
4076  REM ********************
4077  REM
4080  PRINT D$"OPENPOWW/"TME$BASEHEIGHT$
4090  PRINT D$"WRITEPOWW/"TME$BASEHEIGHT$
```

```
4110  PRINT "APPLE " + DTE$ + " POWER (LOG PLOT)"
4111  PRINT HOUR
4112  PRINT MINUTE
4113  PRINT NMREC
4114  PRINT .01 * P2MN * SCLE
4115  PRINT .01 * P1MAX * SCLE
4116  PRINT .01 * P3MEAN * SCLE
4117  PRINT BASEHEIGHT
4130 ADDR = OUTFILEADDR
4140  GOSUB 4680
4150  PRINT D$"CLOSEPOWW/"TME$BASEHEIGHT$
4152  REM
4153  REM ********************
4154  REM * WRITE CORR FILE *
4155  REM ********************
4156  REM
4160  PRINT D$"OPENCORR/"TME$BASEHEIGHT$
4170  PRINT D$"WRITECORR/"TME$BASEHEIGHT$
4180  PRINT "APPLE " + DTE$ + " CORRELATION TIME (SEC)"
4182  PRINT HOUR
4184  PRINT MINUTE
4186  PRINT NMREC
4188  PRINT .05 * C2MN * SCLE
4190  PRINT .05 * C1MAX * SCLE
4192  PRINT .05 * C3MEAN * SCLE
4193  PRINT BASEHEIGHT
4194 ADDR = OUTFILEADDR + 2
4198  GOSUB 4680
4200  PRINT D$"CLOSECORR/"TME$BASEHEIGHT$
4210  REM
4212  REM ********************
4214  REM * WRITE VELL FILE *
4216  REM ********************
4218  REM
4240  PRINT D$"OPENVELL/"TME$BASEHEIGHT$
4250  PRINT D$"WRITEVELL/"TME$BASEHEIGHT$
4260  PRINT "APPLE " + DTE$ + " VELOCITIES (M/S)"
4262  PRINT HOUR
4264  PRINT MINUTE
4266  PRINT NMREC
4268  PRINT .01 * V2MN * SCLE
4270  PRINT .01 * V1MAX * SCLE
4272  PRINT .01 * V3MEAN * SCLE
4273  PRINT BASEHEIGHT
4274 ADDR = OUTFILEADDR + 4
4278  GOSUB 4680
4280  PRINT D$"CLOSEVELL/"TME$BASEHEIGHT$
4310  REM
4311  REM *****************
4312  REM *PRINT SURVEY OF*
4313  REM *  VALID DATA   *
4314  REM *****************
4315  REM
4316  FOR HEIGHT = 0 TO 19
4318 : PRINT HEIGHT;" ";NUMFILES * 2;" ";ENOUGHPOWER(HEIGHT);" ";GOODSHAP
     E(HEIGHT)
```

```
4319  NEXT HEIGHT
4320  END
4325  REM
4327  REM ************
4330  REM *GET HEADER*
4335  REM ************
4340  REM
4350  INPUT "MONTH: ";MNTH$
4360  INPUT "DAY: ";DAY$
4370  INPUT "YEAR: ";YEAR$
4380 DTE$ = MNTH$ + " " + DAY$ + "." + YEAR$
4390  INPUT "HOUR: ";HOUR
4400  INPUT "MINUTE: ";MINUTE
4410  INPUT "NUMBER OF RECORDS: ";NMREC
4415  RETURN
4420  REM
4425  REM **********
4430  REM *MAX, MIN*
4435  REM *AND MEAN*
4437  REM **********
4440  REM
4442  REM  INITIALIZE VARIABLES
4444  REM
4450 P1MAX =  - 32767
4451 P2MN = 32767
4452 P3MEAN = 0
4453 P4COUNT = 0
4454 C1MAX = 0
4455 C2MN = 32767
4456 C3MEAN = 0
4457 C4COUNT = 0
4458 V1MAX =  - 32767
4459 V2MN = 32767
4460 V3MEAN = 0
4461 V4COUNT = 0
4463  REM
4465  REM  FOR ALL DATA
4467  REM
4480  FOR I = OUTFILEADDR TO OUTFILEADDR + NU * 120 - 6 STEP 6
4490 :POW =  PEEK (I) +  PEEK (I + 1) * 256
4500 :CRRTIME =  PEEK (I + 2) +  PEEK (I + 3) * 256
4510 :VEL =  PEEK (I + 4) +  PEEK (I + 5) * 256
4520 : IF POW = 32768 THEN 4650
4530 : IF POW > 32768 THEN POW = POW - 65536
4540 : IF POW > P1MAX THEN P1MAX = POW
4550 : IF POW < P2MN THEN P2MN = POW
4560 :P3MEAN = P3MEAN + POW
4570 :P4COUNT = P4COUNT + 1
4575 : IF CRRTIME = 32768 THEN 4650
4580 : IF CRRTIME > C1MAX THEN C1MAX = CRRTIME
4590 : IF CRRTIME < C2MN THEN C2MN = CRRTIME
4600 :C3MEAN = C3MEAN + CRRTIME
4605 :C4COUNT = C4COUNT + 1
4607 : IF VEL = 32768 THEN 4650
4610 : IF VEL > 32768 THEN VEL = VEL - 65536
4620 : IF VEL > V1MAX THEN V1MAX = VEL
```

```
4630 : IF VEL < V2MN THEN V2MN = VEL
4640 :V3MEAN = V3MEAN + VEL
4645 :V4COUNT = V4COUNT + 1
4650  NEXT I
4652  REM
4654  REM  CALCULATE MEAN
4656  REM
4660 P3MEAN = P3MEAN / P4COUNT
4662 C3MEAN = C3MEAN / C4COUNT
4664 V3MEAN = V3MEAN / V4COUNT
4670  RETURN
4675  REM
4676  REM ************
4677  REM *WRITE DATA*
4678  REM ************
468C
4700    R I = 0 TO 19
4710 : FOR J = 0 TO 119
4720 ::TEMPADDR = ADDR + 6 * I + 120 * J
4730 ::VA =  PEEK (TEMPADDR) +  PEEK (TEMPADDR + 1) * 256
4750 :: IF VA > 32768 THEN VA = VA - 65536
4760 :: PRINT VA * SCLE
4770 : NEXT J
4775  NEXT I
4780  RETURN
```

APPENDIX F

Header File: April 1978

THE DATA ON THIS TAPE ARE PROVIDED IN ACCORDANCE WITH THE MSTRAC PROJECT (MST RADAR COORDINATION) OF THE MIDDLE ATMOSPHERE PROGRAM OF SCOSTEP (SCIENTIFIC COMMITTEE ON SOLAR-TERRESTRIAL PHYSICS). THE REMAINING FILES ON THIS TAPE CONTAIN DATA TAKEN AT THE AERONOMY LABORATORY FIELD STATION, APPROXIMATELY 10 KM NORTH-EAST OF THE UNIVERSITY OF ILLINOIS AT URBANA (40 DEGREES 10'N, 88 DEGREES 10'W). TRANSMITTING FREQUENCY IS 40.92 MHZ AND PEAK TRANSMITTED POWER IS APPROXIMATELY 1250 KW. THE TRANSMITTED PULSE WIDTH IS 20 MICROSECONDS. THE ANTENNA CONSISTS OF 1008 HALF-WAVELENGTH DIPOLE ELEMENTS DIVIDED INTO THREE PARALLEL SECTIONS. THE GROUND WHERE THE ANTENNA IS LOCATED SLOPES 1.5 DEGREES TO THE SOUTH OF EAST, SO THAT THE ON-AXIS ANTENNA POSITION IS OFF BY THE SAME AMOUNT. THE TRANSMITTER AND RECEIVER ARE BOTH CONNECTED TO THE ANTENNA VIA A GAS-FILLED-TUBE TRANSMIT/RECEIVE SWITCH. THE RECEIVER SYSTEM CONSISTS OF A LOW-NOISE BROAD-BAND PREAMPLIFIER, A FILTER AND A SINGLE CONVERSION RECEIVER WITH A BANDWIDTH OF 230 KHZ CENTERED AROUND 40.92 MHZ. THE SIGNAL IS QUADRATURE-PHASE-DETECTED, AND THE TWO COMPONENTS FED THROUGH A MULTIPLEXER AND A 10-BIT ANALOG-TO-DIGITAL CONVERTER WITH A CONVERSION TIME OF 10 MICROSECONDS. DATA PROCESSING IS DONE ON A DIGITAL EQUIPMENT CORPORATION PDP-15 MINI-COMPUTER WITH A 32 K OF CORE MEMORY. PULSE REPETITION FREQUENCY IS 400 HZ AND 20 ALTITUDES ARE SAMPLED. TWENTY-FIVE CONSECUTIVE SAMPLES FROM EACH ALTITUDE RANGE ARE COHERENTLY INTEGRATED SO AS TO GIVE AN INTEGRATED SAMPLE EACH 1/8 SEC. AUTOCORRELATION FUNCTIONS ARE CALCULATED ON-LINE WITH 12 LAGS 1/8 SEC. EACH. THE CORRELATION FUNCTIONS ARE THEN INCOHERENTLY INTEGRATED FOR ONE MINUTE. THESE ONE-MINUTE AVERAGED AUTOCORRELATION FUNCTIONS ARE STORED FOR POST-PROCESSING. SCATTERED POWER AND LINE-OF SIGHT VELOCITY ARE CALCULATED FROM THE AUTOCORRELATION FUNCTION AND STORED ON FLOPPY DISK. THE FILES ON THE FLOPPY DISK WERE USED TO MAKE THIS TAPE.

EACH FILE HAS THE FOLLOWING FORMAT.

TITLE STRING	APRIL 3. POWER(LOGPLOT)
START TIME HOURS	13
START TIME MINUTES	46
NUMBER OF MINUTES IN FILE	120
MINIMUM VALUE	6.28
MAXIMUM VALUE	7.58
AVERAGE VALUE	6.53346485
BASE HEIGHT (KM)	57
DATA DATA DATA DATA ...	635 647 645 695 ....

DATA ARE STORED HEIGHT-BY-HEIGHT, FIRST MINUTE TO LAST. THE FIRST 120 DATA POINTS CORRESPOND TO MINUTES 1 TO 120 FOR THE ALTITUDE (BASE HEIGHT + 1.5) KM. THE NEXT 120 POINTS CORRESPOND TO MINUTES 1 TO 120 FOR (BASE HEIGHT + 3.0) KM. THIS CONTINUES ON FOR EACH HEIGHT UNTIL (BASE HEIGHT + 30) KM IS COMPLETED. THE HEIGHT RESOLUTION IS 1.5 KM. NOTE: DATA STORED ON THIS TAPE ARE 100 TIMES GREATER THAN THE ACTUAL DATA. THIS WAS DONE TO ALLOW AN INTEGER FORMAT WITHOUT LOSS OF PRECISION. SIMPLY DIVIDE EACH DATA VALUE BY

100 TO OBTAIN THE PROPER VALUES (POWER-BELS, VELOCITY-M/S).
ADDITIONAL INFORMATION WHICH MAY PROVE USEFUL.

LABEL=(,NL)
DCB=(RECFM=FB,LRECL=80,BLKSIZE=4000)
EBCDIC
9 TRACK
1600 BPI
129 DATA FILES

QUESTIONS CONCERNING THIS DATA, AND REQUESTS FOR ADDITIONAL DATA SHOULD BE MADE TO PROF. SIDNEY A. BOWHILL, DIRECTOR, AERONOMY LABORATORY, DEPARTMENT OF ELECTRICAL ENGINEERING, UNIVERSITY OF ILLINOIS, 1406 W. GREEN STREET, URBANA, ILLINOIS 61801 USA. THE REMAINDER OF THIS FILE CONTAINS A MENU OF THE TAPE.

FILE	DATA	STARTTIME	DATE
1	POWER	1346	4-3-78
2	VELOCITY	1346	4-3-78
3	POWER	950	4-4-78
4	POWER	1158	4-4-78
5	POWER	1406	4-4-78
6	VELOCITY	950	4-4-78
7	VELOCITY	1158	4-4-78
8	VELOCITY	1406	4-4-78
9	POWER	1206	4-7-78
10	VELOCITY	1206	4-7-78
11	POWER	517	4-10-78
12	POWER	717	4-10-78
13	POWER	917	4-10-78
14	POWER	1141	4-10-78
15	POWER	1632	4-10-78
16	VELOCITY	517	4-10-78
17	VELOCITY	717	4-10-78
18	VELOCITY	917	4-10-78
19	VELOCITY	1141	4-10-78
20	VELOCITY	1632	4-10-78
21	POWER	509	4-11-78
22	POWER	800	4-11-78
23	POWER	1000	4-11-78
24	VELOCITY	509	4-11-78
25	VELOCITY	800	4-11-78
26	VELOCITY	1000	4-11-78
27	POWER	504	4-12-78
28	POWER	704	4-12-78
29	POWER	904	4-12-78
30	POWER	1124	4-12-78
31	POWER	1324	4-12-78
32	POWER	1524	4-12-78
33	POWER	1742	4-12-78
34	VELOCITY	504	4-12-78
35	VELOCITY	704	4-12-78
36	VELOCITY	1124	4-12-78

37	VELOCITY	1324	4-12-78
38	VELOCITY	1524	4-12-78
39	VELOCITY	1742	4-12-78
40	POWER	514	4-13-78
41	POWER	714	4-13-78
42	POWER	1016	4-13-78
43	POWER	1216	4-13-78
44	POWER	1416	4-13-78
45	POWER	1655	4-13-78
46	VELOCITY	514	4-13-78
47	VELOCITY	714	4-13-78
48	VELOCITY	1216	4-13-78
49	VELOCITY	1416	4-13-78
50	VELOCITY	1655	4-13-78
51	POWER	537	4-14-78
52	POWER	737	4-14-78
53	POWER	937	4-14-78
54	POWER	1151	4-14-78
55	POWER	1351	4-14-78
56	POWER	1551	4-14-78
57	VELOCITY	537	4-14-78
58	VELOCITY	737	4-14-78
59	VELOCITY	937	4-14-78
60	VELOCITY	1151	4-14-78
61	VELOCITY	1351	4-14-78
62	VELOCITY	1551	4-14-78
63	POWER	1221	4-18-78
64	POWER	1421	4-18-78
65	POWER	1621	4-18-78
66	VELOCITY	1221	4-18-78
67	VELOCITY	1421	4-18-78
68	VELOCITY	1621	4-18-78
69	POWER	504	4-19-78
70	POWER	704	4-19-78
71	POWER	904	4-19-78
72	POWER	1113	4-19-78
73	POWER	1313	4-19-78
74	POWER	1513	4-19-78
75	POWER	1726	4-19-78
76	VELOCITY	504	4-19-78
77	VELOCITY	704	4-19-78
78	VELOCITY	904	4-19-78
79	VELOCITY	1113	4-19-78
80	VELOCITY	1313	4-19-78
81	VELOCITY	1513	4-19-78
82	VELOCITY	1726	4-19-78
83	POWER	518	4-20-78
84	POWER	718	4-20-78
85	POWER	918	4-20-78
86	POWER	1156	4-20-78
87	POWER	1356	4-20-78
88	POWER	1556	4-20-78
89	VELOCITY	518	4-20-78
90	VELOCITY	718	4-20-78

91	VELOCITY	918	4-20-78
92	VELOCITY	1156	4-20-78
93	VELOCITY	1356	4-20-78
94	VELOCITY	1556	4-20-78
95	POWER	454	4-21-78
96	POWER	654	4-21-78
97	POWER	854	4-21-78
98	POWER	1131	4-21-78
99	POWER	1331	4-21-78
100	POWER	1531	4-21-78
101	POWER	1741	4-21-78
102	VELOCITY	454	4-21-78
103	VELOCITY	654	4-21-78
104	VELOCITY	854	4-21-78
105	VELOCITY	1131	4-21-78
106	VELOCITY	1331	4-21-78
107	VELOCITY	1531	4-21-78
108	VELOCITY	1741	4-21-78
109	POWER	441	4-24-78
110	POWER	641	4-24-78
111	POWER	841	4-24-78
112	POWER	1109	4-24-78
113	POWER	1509	4-24-78
114	POWER	1719	4-24-78
115	VELOCITY	441	4-24-78
116	VELOCITY	641	4-24-78
117	VELOCITY	841	4-24-78
118	VELOCITY	1109	4-24-78
119	VELOCITY	1309	4-24-78
120	VELOCITY	1509	4-24-78
121	VELOCITY	1719	4-24-78
122	POWER	444	4-25-78
123	POWER	644	4-25-78
124	POWER	844	4-25-78
125	POWER	1054	4-25-78
126	VELOCITY	444	4-25-78
127	VELOCITY	644	4-25-78
128	VELOCITY	844	4-25-78
129	VELOCITY	1054	4-25-78

APPENDIX G
Listing of Reformatting Program

```
1  DIM A(2600),B(7)
10  REM  PROGRAM TO REFORMAT TEXT FILES
20  REM  WITH RECORD LENGTHS OF 6 OR
30  REM  LESS, TO TEXT FILES WITH
40  REM  RECORDS LENGTH 80.
42  ONERR  GOTO 220
45  PRINT "ENTER NAME OF FILE TO BE REFORMATED"
50  INPUT I$
60 O$ = "A" + I$
70  PRINT "REFORMATTED FILENAME IS ";O$
80 D$ =  CHR$ (4)
95  PRINT D$"MON C,I,O"
130  INPUT "INSERT SOURCE DISK...";F$
140  PRINT D$;"OPEN";I$
150  PRINT D$;"READ";I$
155 X =  FRE (0)
160  INPUT L$
162  FOR I = 1 TO 7
163  INPUT B(I)
164  NEXT I
170 I = 1
180  INPUT A$
190  IF  LEN (A$) > 6 THEN 222
200 A(I) =  VAL (A$)
210 I = I + 1: GOTO 180
220  IF  PEEK (222) >  < 5 THEN 450
222 LIMIT = I - 1
230  PRINT D$;"CLOSE";I$
235  INPUT "INSERT DESTINATION DISK...";F$
240  PRINT D$;"OPEN";O$
250  PRINT D$;"WRITE";O$
260  PRINT L$
270  FOR I = 1 TO 7
280  PRINT B(I)
290  NEXT I
300 I = 0
310 J = 1
320  PRINT  RIGHT$ ("       " +  STR$ (A(I + J)),7);
330  IF J = 11 THEN  PRINT "   "
340  IF (I + J) = LIMIT THEN 370
350 J = J + 1: IF J < 12 THEN 320
360 I = I + 11: GOTO 310
370 T = J + 1
380  IF J = 12 THEN 420
390  PRINT "       ";
400  IF T = 11 THEN  PRINT "   "
410 T = T + 1: IF T < 12 THEN 380
420  PRINT D$;"CLOSE";O$
430  PRINT "REFORMATTED FILENAME IS"
440  PRINT C$
442 X =  FRE (0)
445  GOTO 45
450  END
```

REFERENCES

Bowhill, S. A. [1983], Review of correlation techniques, Handbook for MAP Volume 9, 521-526, Edited by S. A. Bowhill and Belva Edwards, SCOSTEP Secretariat, Univ. IL., 1406 W. Green St., Urbana, IL. 61801.

Gibbs, K. P. and S. A. Bowhill [1983], An investigation of turbulent scatter from the mesosphere as observed by coherent-scatter radar, Aeron. Rep. No. 110, Aeron. Lab., Dept. Elec. Eng., Univ. IL., Urbana, IL. 61801.

Lyons, G. B. [1981], Fig-FORTH Release 1.0 May, 1979, FORTH Interest Group, Box 1105, San Carlos, CA. 94070.

Roth, P. R. [1982] (Ed.), Research in Aeronomy, October 1, 1981 - March 31, 1982, Prog. Rep. 82-1, Aeron. Lab. Dept. Elec. Eng., Univ. IL., Urbana, IL. 61801.

Roth, P. R. [1983] (Ed.), Research in Aeronomy, October 1, 1982 - March 31, 1983, Prog. Rep. 83-1, 33-35, Aeron. Lab. Dept. Elec. Eng., Univ. IL., Urbana, IL. 61801.

Scanlon, L. J. [1980], 6502 Software Design, Howard W. Sams and Co., Inc., Indianapolis, IN. 46206.

Scanlon, L. J. [1982], FORTH Programming, Howard W. Sams and Co., Inc., Indianapolis, IN. 46266.

C-2